Escuela
Politécnica
Superior

Sistema de visión egocéntrica de reconocimiento de objetos cotidianos para el apoyo a personas con movilidad reducida

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Gilberto Jesús Brito
Tutor/es:
Francisco Flórez Revuelta
Francisco Javier Esclapés

Junio 2020

Universitat d'Alacant
Universidad de Alicante

Preámbulo

Este proyecto se ha realizado para crear una aplicación de un sistema de visión egocéntrica que mediante reconocimiento de objetos apoyase a las personas con movilidad reducida.

En segundo lugar, este trabajo de fin de grado forma parte del proyecto en el que se desarrolla un exoguante de bajo coste por parte de [artefactos] que es un proyecto de investigación multidisciplinar que busca generar soluciones alternativas y accesibles en código abierto para mejora de la autonomía y calidad de vida de personas con diversidad funcional.

Y por último, destacar que supone el punto de partida para la inclusión de técnicas de visión por computador en entornos de visión egocéntrica en el proyecto del exoguante.

Agradecimientos

Gracias a mi tutor, Paco Flórez, por haberme acompañado durante todo el proyecto marcándome el camino a seguir, también por haberme guiado en las tareas que no sabía solucionar durante el desarrollo del trabajo, y por último, por dotarme de las herramientas necesarias para la elaboración del mismo. Gracias también a mi cotutor Javier Esclapés por haberme hecho conocer este proyecto y la iniciativa de [artefactos]. Gracias a Pau por haberme ayudado con los distintos problemas que han surgido durante el proyecto, compartido conmigo recursos, y demás.

Por otra parte, me gustaría agradecer a mis amigos y compañeros que han sido un apoyo fundamental durante todo el transcurso del grado.

También me gustaría acordarme de mis amigos que la vida ha convertido en mi familia.

Y por último, gracias mamá por haberme apoyado todo este tiempo, aún quedan muchísimos retos que superar y logros por alcanzar.

Sin desviarse de la norma
el progreso es imposible.

Frank Zappa.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

Durante este apartado se explicarán las razones por las que se ha realizado este proyecto. También se mencionará y razonará el alcance del trabajo, los objetivos planteados y la propuesta para dar solución al problema específico que daba razón al mismo.

1.1 Motivación

Este trabajo de fin de grado se realiza como complemento a otros trabajos de fin de grado y de fin de máster que han elaborado un prototipo de exoguante[1] de bajo coste para el proyecto de [artefactos][2].

[artefactos] es proyecto de investigación multidisciplinar que busca generar soluciones alternativas y accesibles en código abierto para la autonomía y calidad de vida de personas con diversidad funcional, mediante la confluencia del diseño colaborativo, la tecnología y la innovación.

Por ello, la principal razón para elaborar este trabajo de fin de grado es la de crear un sistema que mediante la percepción de objetos cotidianos pudiera apoyar a personas con diversidad funcional. Principalmente, mediante la decisión del tipo de agarre que se deba adaptar por el mecanismo de actuación del exoguante y de esta manera facilitar el uso del prototipo al usuario.

1.2 Resumen del trabajo

El trabajo ha consistido en el entrenamiento de una red neuronal para la detección del objeto que aparezca en el campo de visión de la cámara desde una perspectiva egocéntrica, es decir, desde el punto de vista del usuario.

Para realizar esta tarea se ha tenido que realizar un estudio de los diferentes conjuntos de datos de visión egocéntrica existentes y así determinar cuál se usaría en este proyecto. Finalmente se ha optado por el uso del conjunto de datos EPIC-KITCHENS[3] dataset.

Después de establecer con qué conjunto de datos realizaríamos el entrenamiento de los algoritmos, el conjunto de datos ha sido explorado para entender cómo estaba estructurada la información. A continuación de la fase exploratoria, se ha realizado un tratamiento de esta información que nos permitiese extraer la información procesada, o sea, obtener el número de clases, qué clases tienen más ocurrencias dentro de nuestro conjunto de datos, etc. Y finalmente, se ha reestructurado la información para que fuese posible el entrenamiento de los algoritmos de detección de objetos.

Por otra parte, se ha realizado una comparativa de las diferentes alternativas de arquitecturas de redes neuronales que se suelen emplear para tareas de detección de objetos. Con respecto a esto, se buscaba que fuera tanto eficiente en el tiempo de inferencia como que tuviese una precisión aceptable con respecto a los resultados arrojados por el equipo de EPIC-KITCHENS, esta precisión ha sido

[1] `https://www.artefactos.org/tag/exoguante/` Accedido el 02/06/2020
[2] `https://www.artefactos.org/` Accedido el 02/06/2020
[3] `https://epic-kitchens.github.io/2020` Accedido el 02/06/2020

medida utilizando la métrica mAP (mean Average Precision)[4].

La consideración de que el tiempo de inferencia fuese bajo es importante porque se busca que cuando se integre todo el sistema, los requerimientos técnicos del equipo que lanzará la inferencia no sean muy elevados debido a que el proyecto pretende seguir la filosofía de producir un prototipo de bajo coste.

1.3 Objetivos

En este apartado se listarán y explicarán los objetivos generales, los objetivos específicos y la propuesta realizada para conseguir el cumplimiento de los objetivos marcados.

1.3.1 Objetivos generales y específicos

El objetivo principal de este proyecto como se ha mencionado anteriormente será la detección de los objetos cotidianos, empleados en la cocina, mediante técnicas de visión por computador. Específicamente técnicas de Deep Learning[5]. Establecido esto, la siguiente lista reúne los objetivos generales y los objetivos específicos dentro de cada uno de los generales.

- Analizar el conjunto de datos sobre el que trabajaremos para así determinar qué procesamiento debemos llevar a cabo.

  Realizar una visualización de la información, pudiendo ser mediante gráficas, visualización de las imágenes anotadas, etc.

- Filtrar la información eliminando del conjunto de datos la información que no esté correctamente etiquetada, sea nula, o simplemente que no sea relevante para la consecución del objetivo principal.

  Elaborar un filtro que nos permita pulir la información y obtener un dato con mayor valor.

  Entender qué dato consideramos relevante.

- Preparar la información para que sea posible entrenar los algoritmos candidatos sobre la información reestructurada.

- Entrenamiento de diferentes algoritmos de aprendizaje profundo para analizar cuál se adaptaría mejor a nuestro problema.

  Crear un entorno que permita el entrenamiento de los algoritmos.

  Modificar el entrenamiento para que se adapte a nuestras necesidades específicas, estas modificaciones podrían estar relacionadas con:

  Número de épocas.

  La separación de los datos para validar que el algoritmo está aprendiendo a reconocer nuestros objetos.

  Modificación del learning rate[6].

  La utilización o no de aumentado de datos.

[4] `https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173` Accedido el 02/06/2020

[5] `https://www.bbvaopenmind.com/tecnologia/mundo-digital/que-es-el-aprendijaze-profundo/` Accedido el 02/06/2020

[6] `https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10` Accedido el 02/06/2020

Descongelar las capas más profundas de las arquitecturas de las redes neuronales[7] para entrenar de cero.

La creación periódica de puntos de guardado según un número determinado de épocas.

La especificación de la ruta de las anotaciones sobre las imágenes a entrenar.

- Encontrar diferentes implementaciones de los algoritmos candidatos a ser analizados y utilizados en nuestro proyecto.

  La comprobación del correcto funcionamiento de las implementaciones que se usarán.

- Analizar los resultados de la métrica usada mAP para los diferentes algoritmos y compararlos con los resultados arrojados en el artículo publicado por los autores del conjunto de datos.

  Encontrar implementaciones del calculo de esta métrica.

  Visualizar el resultado para cada arquitectura probada.

- Visualizar en video y/o en directo la inferencia del modelo de aprendizaje profundo sobre los objetos que se prenteden detectar.

  Elaboración de una herramienta para la visualización de la inferencia.

[7] `https://towardsdatascience.com/introduction-to-artificial-neural-networks-ann-1aea15775ef9` Accedido el 02/06/2020

2 Estado del arte

Este apartado tiene el fin de poner en relieve el estado actual de las técnicas y campos asociados a este proyecto.

2.1 Tecnología asistencial

En primer lugar, debemos definir qué es la tecnología asistencial. La tecnología asistencial es cualquier artefacto *software* o *hardware* diseñado para ayudar a las personas con diversidad funcional, incrementando, manteniendo o mejorando sus capacidades funcionales. Estos artefactos están pensados principalmente para que estos usuarios puedan mejorar su desempeño en actividades cotidianas que sean de dificultad para ellos.

La tecnología asistencial puede cubrir diferentes discapacidades, por ejemplo, los miembros de la asociación ATIA (Assistive Technology Industry Association)[1] intentan aportar tecnologías para las siguientes:

- Ceguera o ceguera parcial.
- Sordera o sordera parcial.
- Problemas de accesibilidad a los ordenadores.
- Problemas de comunicación.
- Problemas de movilidad.
- Discapacidades cognitivas.

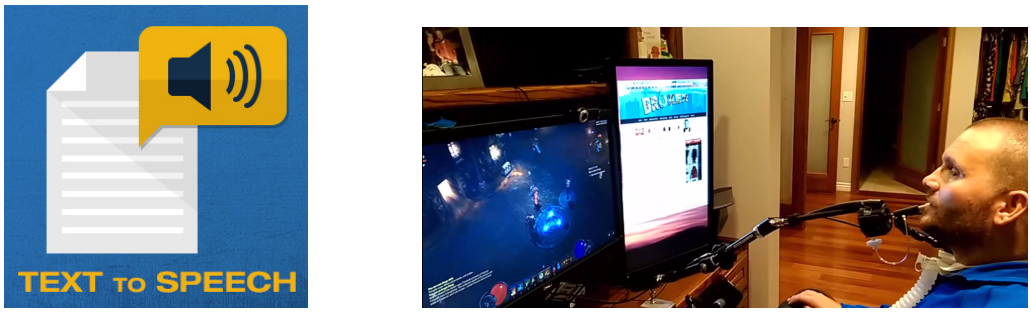


Figura 2.1: Ejemplo de dispositivos asistenciales

2.1.1 Tecnología asistencial para el agarre

El proyecto del exoguante se incluye dentro de esta categoría de tecnología asistencial.

[1] `https://www.atia.org/home/at-resources/what-is-at/#what-is-assistive-technology` Accedido el 02/06/2020

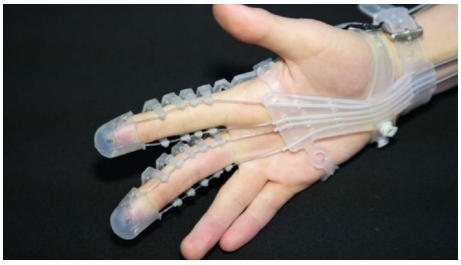
Principalmente se ha inspirado en (In y cols., 2015), (Randazzo y cols., 2018) y en la mejora que supuso el incluir un sistema de percepción en el caso de (Kang y cols., 2019).

En el caso de (In y cols., 2015) se elaboró un prototipo como alternativa a exoesqueletos tradicionales como pudiera ser (Catalán y cols., 2017) para conseguir un instrumento más ligero. En el caso de (In y cols., 2015) el mecanismo de actuación se realiza mediante señales EMG[2] de un musculo alternativo al disfuncional.

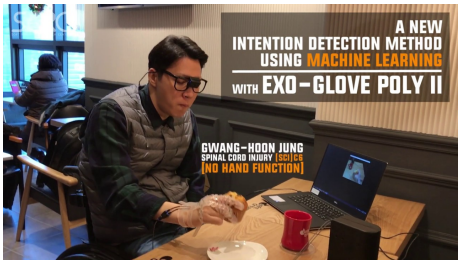
Por otra parte, aparte de ser un instrumento asistencial más ligero, se tornaba más accesible por el hecho de que el mantenimiento higiénico del prototipo es más simple que el de un exoesqueleto. También, el precio es más reducido, se puede producir a gran escala, y además, la facilidad de portarlo durante las actividades cotidianas es mayor.

Posteriormente, se desarrolló (Kang y cols., 2019) en donde la actuación también se basaba en la percepción de un sistema de visión egocéntrica, y de la predicción de la intención del usuario basado en un algoritmo de aprendizaje automático.

Ver Figura 2.2 donde se muestran ejemplos gráficos de estos proyectos.



(a) (In y cols., 2015)



(b) (Kang y cols., 2019)

(c) (Catalán y cols., 2017)

(d) (Randazzo y cols., 2018)

Figura 2.2: Ejemplos de prototipos asistenciales para el agarre

[2] `http://www.encuentros.uma.es/encuentros53/aplicaciones.html` Accedido el 02/06/2020

2.1.2 Tecnología asistencial y aprendizaje automático

Aparte de los prototipos y proyectos ya citados previamente, en este apartado se mencionarán otros que hagan principal o complementariamente uso de técnicas de aprendizaje automático.

Uno de estos proyectos es (Fang y cols., 2017) donde mediante técnicas de aprendizaje profundo y redes neuronales recurrentes se elaboró un sistema que tradujese de lenguaje de signos americano (ASL) a voz. Consiguiendo una media de 94,5% de precisión a nivel de palabra y una media del 8,2% de ratio de error al traducir palabras en lenguaje de signos americano que no habían sido vistas previamente. Como dispositivos utilizaron un Leap Motion para capturar el movimiento de las manos y unas Microsoft Hololens para tareas de realidad aumentada. Ver Figura 2.3 para ver ejemplo de funcionamiento de DeepASL.

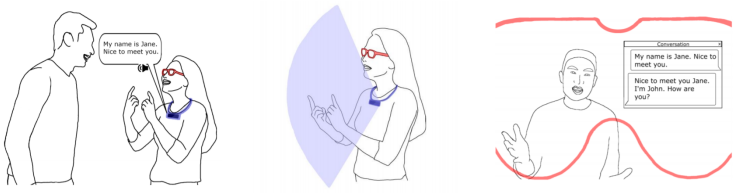


Figura 2.3: Extracto de (Fang y cols., 2017)

Un uso diferente de las técnicas de aprendizaje automático en las tecnologías asistenciales es el caso de los diferentes proyectos de silla de ruedas autónomas, controladas mediante la vista, etc. Como se menciona en (Desai y cols., 2017)

Por otra parte, existen multitud de proyectos que están siendo amparados bajo la iniciativa de Microsoft AI for Accesibility. Dentro de esta iniciativa está iTherapy[3] que ha desarrollado un software que permite a niños del espectro autista a mejorar sus habilidades de comunicación, incluye reconocimiento facial, tecnología de texto a voz, etc.

Por último, cabe mencionar el uso de sensores y registros biométricos para registrar la actividad de los usuarios. Esta información puede ser posteriormente procesada por algoritmos de aprendizaje automático para realizar tareas que pretendan asistir a usuarios de diversidad funcional, ancianos, etc.

2.2 Visión egocéntrica

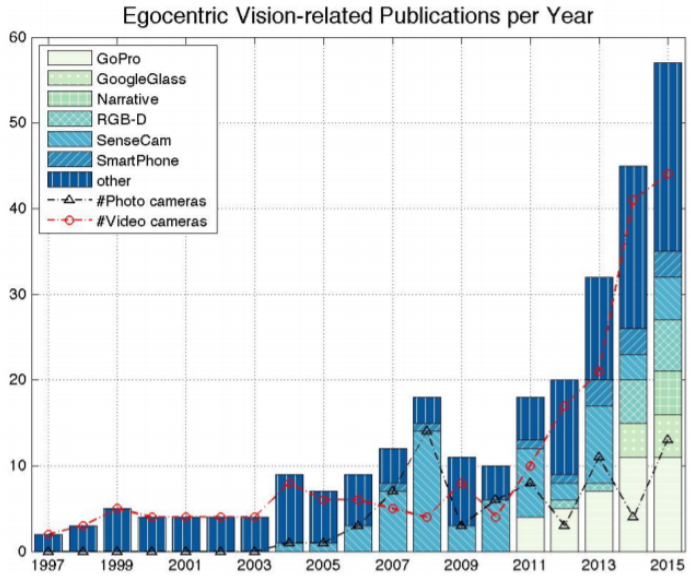
La visión egocéntrica[4] es un campo de la visión artificial que se encarga de analizar imágenes y vídeos capturados mediante cámaras *wearable*. Normalmente estas cámaras están sujetas al pecho o a la cabeza del usuario. También existen alternativas con formato de gafas.

2.2.1 Dispositivos de visión egocéntrica

Como se ha mencionado existen diferentes tipos. Hasta 2015 el uso de las diferentes opciones era el de la Figura 2.4.

Existen dos tipos de cámaras:

[3] `https://www.itherapyllc.com/` Accedido el 02/06/2020
[4] `http://www.ub.edu/cvub/egocentric-vision/` Accedido el 02/06/2020



(a) Número de artículos de investigación publicados según el tipo de cámara de visión egocéntrica

Figura 2.4: Extracto de (Bolaños y cols., 2015)

- Cámaras de vídeo: Estas cámaras tienen una alta resolución temporal (de 25 a 60 imágenes por segundo). Muy útiles para grabar actividades como cocinar, la realización de deportes, etc. Dentro de esta categoría se encuentran las cámaras GoPro y Google Glass.

(a) GoPro Hero 8

(b) Google Glass

Figura 2.5: Cámaras de vídeo egocéntrico

- Cámaras de fotos: Estas cámaras por el contrario tienen una baja resolución temporal (de 2 a 3 imágenes por minuto). Son más útiles para capturar información sobre periodos de tiempo prolongados. Narrative Clip 2 es de este tipo.

2.3 Aprendizaje automático

Es importante la definición de aprendizaje automático puesto que los siguientes apartados forman parte de este campo. Se puede definir como una rama de la inteligencia artificial cuyo objetivo es desarrollar algoritmos o heurísticas que aprendan en base a unos datos y generen instrucciones. Esto es lo contrario a la programación tradicional que el programador se encarga de que mediante instrucciones se genera un comportamiento.

(a) Narrative Clip 2

Figura 2.6: Cámara de fotos de visión egocéntrica

2.3.1 Aprendizaje profundo

El aprendizaje profundo es s un subconjunto de los algoritmos de aprendizaje automático o machine learning.

Lo que hace que se pueda definir el deep learning como un subconjunto del machine learning es que los algoritmos de deep learning suelen estar compuestos por varias capas con unidades de procesamiento que extrae y transforma variables. Cada una de estas capas usa la información de las capas anteriores como entrada. Y la capa de más alto nivel tienen un nivel de abstracción más alto que las más profundas.

Estos algoritmos se emplean para numerosos campos, los más relevantes son:

- Visión por computador.
- Procesamiento del lenguaje natural.
- Procesamiento de señales.
- Reconocimiento del habla.

Nosotros durante este proyecto nos centraremos en el uso de técnicas de deep learning orientadas a visión por computador, concretamente la detección de objetos.

2.4 Visión por computador

Este será el campo concreto donde se situaría el proyecto. La visión por computador es un campo que pretende extraer información de imágenes o de una secuencia de imágenes.

Algunas tareas que están englobados dentro de la visión por computador son las siguientes:

- Clasificación de imágenes
- Detección de objetos
- Estimación de la pose humana
- Reconstrucción de escenas

2.4.1 Detección de objetos

Se deben definir unos conceptos previos que se pueden observar en la Figura 2.7:

- **Clasificación de objetos**: Esta técnica predice cual es la probabilidad de que diferentes categorías de objetos estén en una imagen. Solo una categoría por imagen.
- **Localización de objetos**: Este método predice cual es la probabilidad de que un objeto esté en una imagen junto a su localización.
- **Detección de objetos**: Los métodos anteriores solamente predicen las probabilidades para solo un objeto, en cambio la detección de objetos incluye la clasificación y localización de más de un objeto por imagen. Al detectar un objeto se dibuja una caja llamada *bounding box*.
- **Segmentación de instancias**: Al contrario que la detección de objetos, la segmentación de instancias de objetos extrae la forma exacta del objeto en la imagen.

Cada clase se clasifica basada en sus características. Cada imagen tiene unas características y bordes únicos. En el caso de la detección de objetos estas características son las que aprende el algoritmo, para así posteriormente predecirlas.

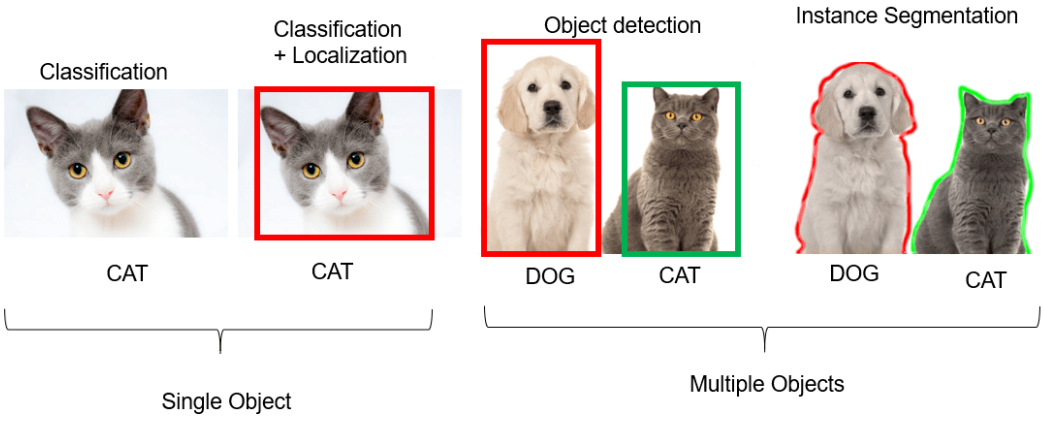


Figura 2.7: Diferencias entre localización, clasificación, detección y segmentación

Existen diferentes aproximaciones a este problema. Las aproximaciones que forman parte del conjunto de técnicas de machine learning son:

- **Viola-Jones**. (Viola y Jones, 2001). basado en características Haar.
- **Scale-invariant feature transform**. (Lindeberg, 2012).
- **Histograma de gradientes orientadas**. (Dalal y Triggs, 2005).

Las aproximaciones más relevantes que pertenecen al grupo de los algoritmos de deep learning son:

- **R-CNN**. (Girshick y cols., 2013).
- **YOLO**. (Redmon y cols., 2015).
- **Single Shot MultiBox Detector (SSD)**. (Liu y cols., 2015).

Los algoritmos basados en deep learning pueden dividirse en dos grupos:

- **Basados en clasificación**: Existen dos etapas. La primera etapa recoge diferentes regiones de interés (ROI) donde la probabilidad de que el objeto se encuentre sea alta. La segunda etapa aplica una red neuronal convolucional. Es el caso de (Girshick, 2015).
- **Basados en regresión**: No se seleccionan ROIs, se predice las clases y las bounding boxes de la imagen de una sola pasada. Es el caso de (Redmon y cols., 2015).

2.4.2 You Only Look Once

2.4.2.1 Funcionamiento You Only Look Once

En primer lugar, antes de definir cómo está estructurada esta red neuronal, se debe explicar cómo funciona.

YOLO puede predecir la clase del objeto, su bounding box y la probabilidad del objeto dentro de la bounding box. Cada bounding box tiene un centro, ancho, alto, la clase del objeto y la probabilidad de la clase asociada. Como se puede observar en la Figura 2.8.

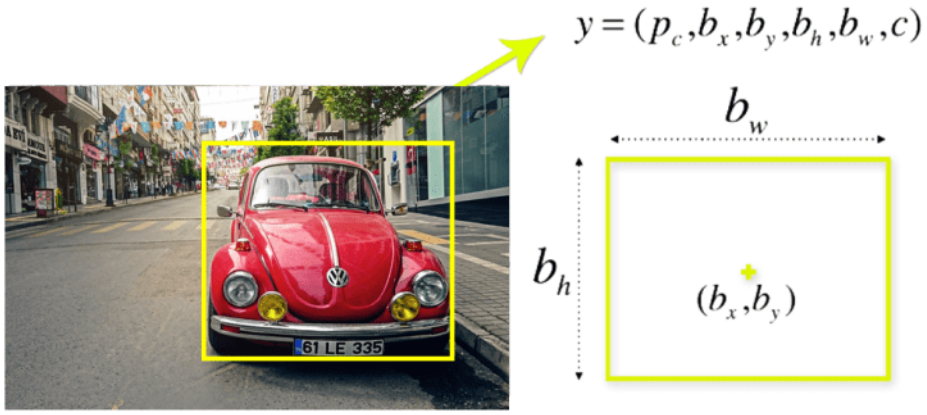


Figura 2.8: Ejemplo de bounding box.

YOLO como se ha mencionado previamente no busca unas ROIs concretas. Realiza por el contrario, lo siguiente: divide la imagen en diferentes celdas, normalmente usando una matriz de 19x19. Cada celda es reponsable de predecir cinco bounding boxes. Al final una imagen tendría 1805 bounding boxes. Ver figura 2.9.

Como no todas las bounding boxes contendrán objetos se filtran usando la probabilidad de la clase del objeto. Utilizando el metodo Non-maximum Supression (NMS) se eliminan las bounding boxes que no queramos y nos quedamos solamente las que tienen mayor probabilidad. Ver Figura 2.10.

2.4.2.2 Arquitectura y evolución de YOLO

Este apartado estudiarán los cambios y evoluciones que ha sufrido YOLO. Esto es interesante realizarlo por el hecho de que existen cuatro versiones en donde se han modificado la arquitectura que existe por debajo de YOLO (Darknet), uso de otras técnicas, entre otros.

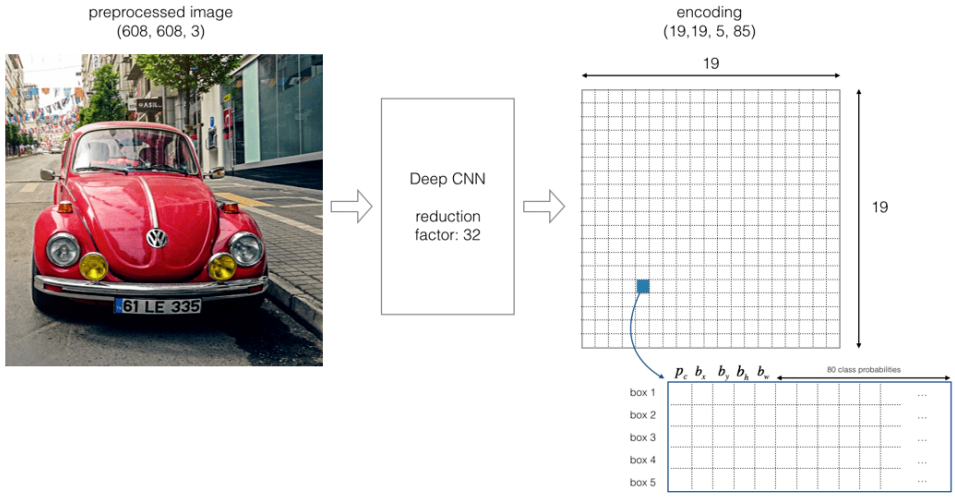


Figura 2.9: Funcionamiento de YOLO.

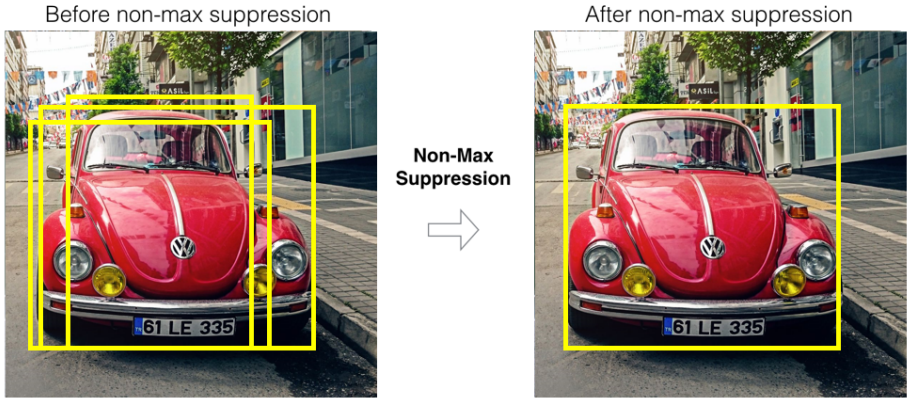


Figura 2.10: Ejemplo al aplicar NMS.

La primera versión de YOLO (Redmon y cols., 2015) usaba de base Darknet(Redmon, 2013–2016). La primera arquitectura estaba formada por 24 capas convolucionales seguidas de dos capas completamente conectadas. Las 24 capas convolucionales se usan para extraer las características y las dos últimas para realizar las predicciones.

Posteriormente, la segunda versión de YOLO (Redmon y Farhadi, 2016) mejoraba la detección de los objetos que estaban cerca en la imagen. Esta versión incluyo *anchor boxes* que son un conjunto de bounding boxes predefinidas con ciertas dimensiones. Se usó Darknet con 19 capas convolucionales y 5 capas de *maxpooling*.

En la tercera versión (Redmon y Farhadi, 2018) se hicieron pequeñas mejoras, incluyendo el hecho de que las bounding boxes fueran predichas a diferentes escalas. En esta versión Darknet pasó de tener 24 capas convolucionales a 53 capas convolucionales. Ver Figura 2.12. Por último, la YOLO v4 no ha sido desarrollada por el autor original de la red. En (Alexey Bochkovskiy, 2020) se verifica cómo influyen nuevos métodos para mejorar el entrenamiento de los algoritmos de detección de objetos

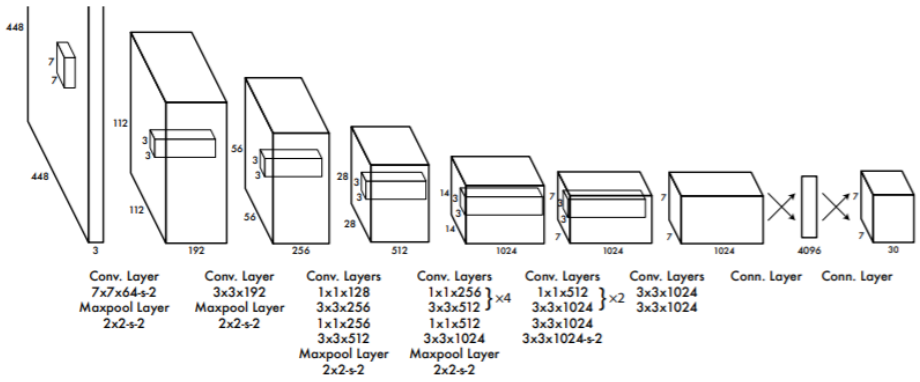


Figura 2.11: Arquitectura YOLO v1 (incluye Darknet24) extraida de (Redmon y cols., 2015).

como *Bag-of-Freebies* y *Bag-of-Specials*. Además emplea EfficientNet como backbone al contrario de su predecesora YOLO v3 que usa como backbone la arquitectura Darknet53.

2.4.3 Otras arquitecturas

En este apartado se tratarán otras arquitecturas para detección de objetos diferentes a YOLO.

2.4.3.1 Faster R-CNN

Esta arquitectura se define en (Ren y cols., 2015). La explicaremos por el hecho de que se usa en (Damen y cols., 2020) para realizar los benchmarks. A diferencia de (Girshick y cols., 2013) y de (Girshick, 2015) no usa un algoritmo de búsqueda selectiva, si no que permite a la red aprender las regiones importantes. Ver Figura 2.13.

2.4.3.2 EfficientNet

Desarrollada esta arquitectura por el equipo de Google Brain, (Tan y Le, 2019). Se usó como backbone para (Tan y cols., 2019). Superó en resultados a modelos de tamaño similar en los benchmarks de diferentes datasets.

La motivación de este diseño viene dada por buscar exponer el proceso de escalado en las arquitecturas de redes convolucionales. EfficientNet establecío que se definiera automáticamente los parámetros de la red convolucional. En (Tan y Le, 2019) se buscaba optimizar la performance mediante un rango libre de profundidad, anchura y resolución mientras se mantuviese dentro de los límites establecidos la memoria y los FLOPs.

2.4.4 Comparativa MSCOCO Dataset

Esta comparativa se realiza bajo este dataset puesto que este conjunto de datos forma parte del estándar en la visión por computador para evaluar la precisión de los detectores de objetos, junto a otros como podría ser el PASCAL VOC.

En la Figura 2.15 podemos observar cómo el EfficientDet (tiene una precisión similar a YOLO v4 consiguiendo menos imágenes por segundo. También podemos observar cómo YOLO v4 mejora con respecto a YOLO v3 un 10% y un 12% de AP y de FPS respectivamente. En esta comparativa se utilizó una tarjeta gráfica NVIDIA Tesla V100 para la inferencia.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 2.12: Darknet53 tabla extraída de (Redmon y Farhadi, 2018)

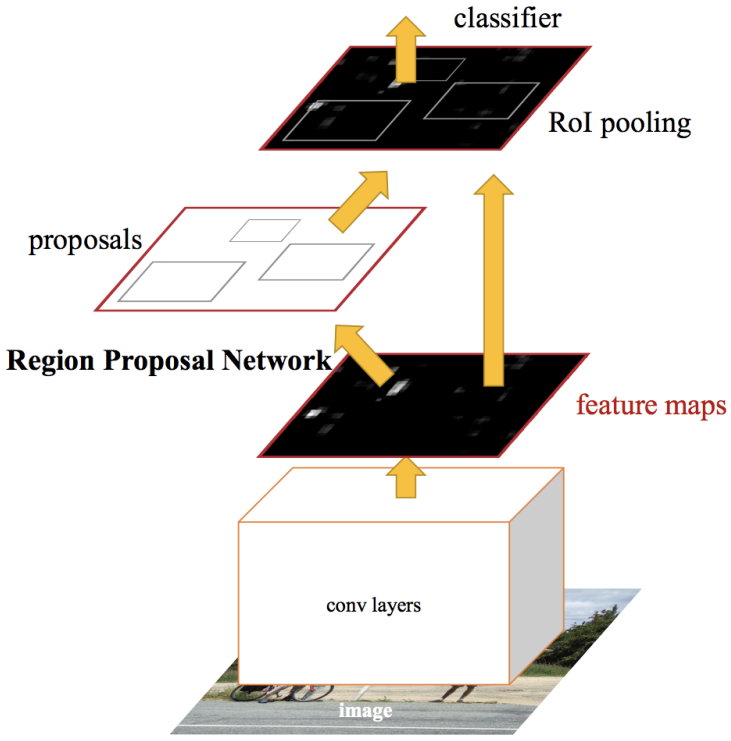


Figura 2.13: Figura extraída de (Ren y cols., 2015).

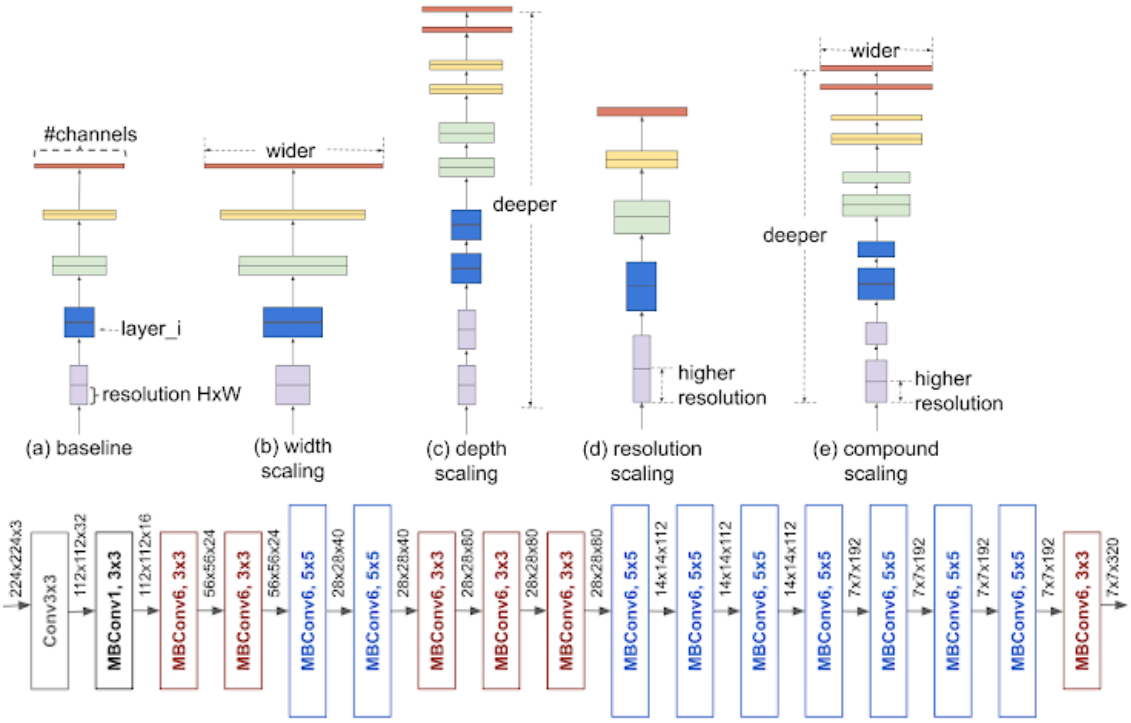


Figura 2.14: Figuras extraídas de (Tan y Le, 2019). Arquitectura EfficientNet.

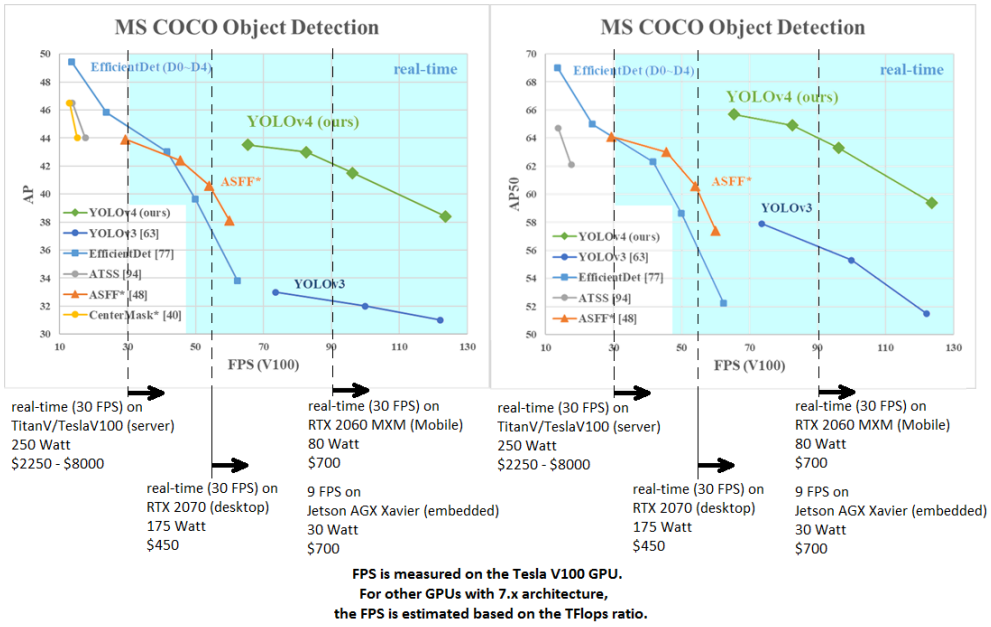


Figura 2.15: Figura extraída de (Alexey Bochkovskiy, 2020).

3 Conjunto de datos

En este punto se tratará el proceso que se ha seguido para la selección del conjunto de datos, análisis del conjunto de datos, procesamiento de los datos hasta dejarlo todo preparado para que la red neuronal pueda procesar su información.

3.1 Datasets relacionados

Existen diversos datasets de visión egocéntrica que se han mencionado como relevantes en (Damen y cols., 2020) y también en (Nguyen y cols., 2016). Por ello, se han elegido los que se han visto como más relevantes y relacionados a nuestro problema.

Por una parte tenemos (Ryoo y Matthies, 2013) que es un dataset compuesto por vídeos de actividades grabados en primera persona. Contiene vídeos de las interacciones entre humanos y el observador. Se usó una cámara GroPro2 situada en cabeza del modelo humanoide. Los vídeos están en una resolución de 320x240 y a 30 FPS. Ver Figura 3.1.

Hay 7 tipos diferentes de actividades, incluyendo 4 actividades positivas hacia el observador, 1 neutral, y 2 negativas.

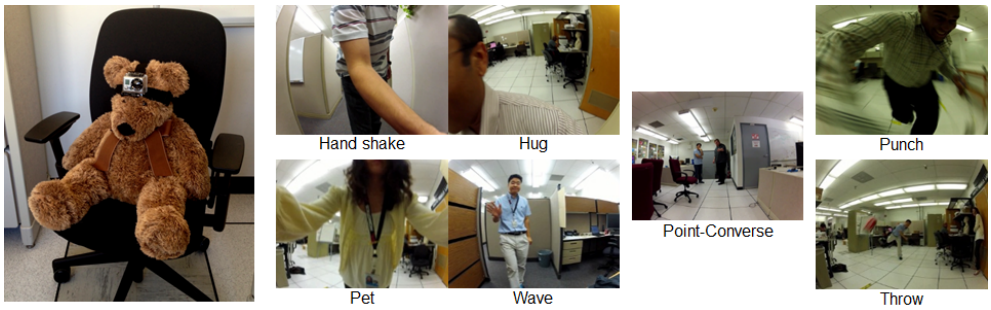


Figura 3.1: Figura extraída de (Ryoo y Matthies, 2013)

En el caso de (Pirsiavash y Ramanan, 2012) también se usó una cámara GoPro. Se sujetó en el pecho del observador. Esta cámara captura bajo una resolución de 1280x960 y a 30fps con 170 grados de visión. Se creo una lista de 18 actividades diarias. Se pudo reunir más de 10 horas de vídeo en primera persona. Ver Figura 3.2.

3.1.1 Estructura EPIC-KITCHENS dataset

En el caso de (Damen y cols., 2020) fue el dataset escogido finalmente para entrenar nuestros algoritmos de detección de objetos. Para la detección de objetos existen varios datasets que se pudieran considerar el estándar, como son MS-COCO, ImageNet y Pascal VOC. Lo que ha marcado la diferencia con respecto a estos conjuntos de datos es que (Damen y cols., 2020) está creado desde una

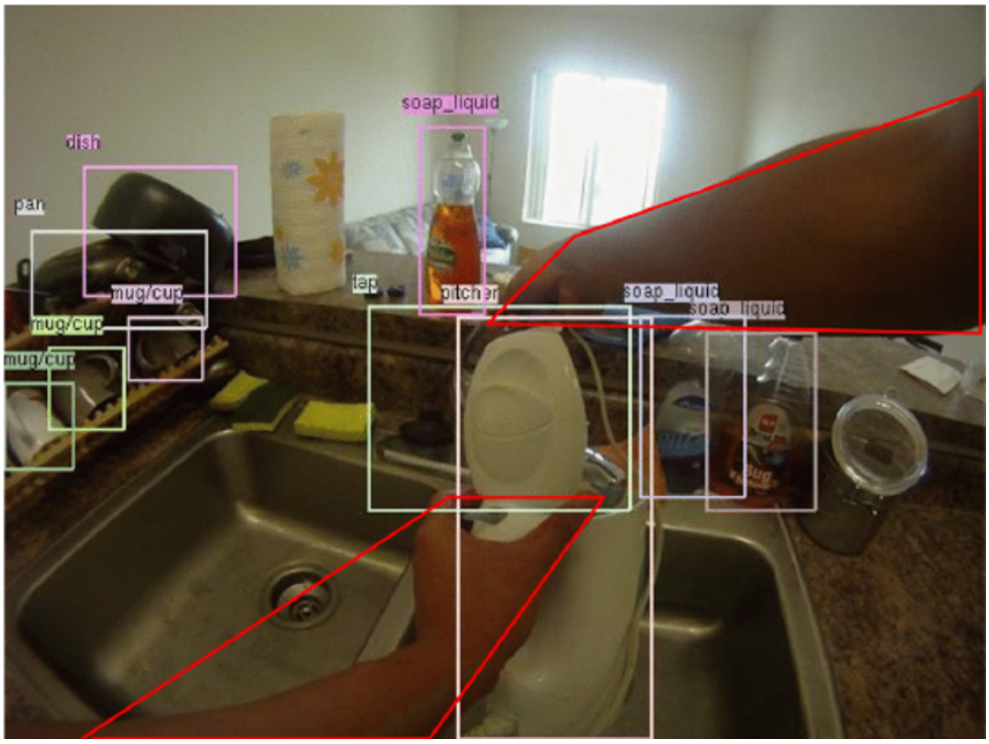


Figura 3.2: Figura extraída de (Pirsiavash y Ramanan, 2012)

perspectiva de primera persona.

En comparación a los datasets de visión egocéntrica previamente nombrados el caso de EPIC-KITCHENS tiene 454,158 bounding boxes de objetos anotados (Ver Figura 3.3), mientras que en el caso de ADL solamente tiene anotado 137,780.

EPIC-KITCHENS ha sido capturado mediante una cámara GoPro situada en la cabeza del observador. La captura se ha realizado empleando una configuración de campo de visión linear a 59.94 FPS y a una resolución de 1920x1080. En algunos casos esto varió, 1% de los vídeos fueron grabados a 1280x720y 0,5% a 1920x144. También 1% a 30 FPS, 1% a 48 FPS y 0.2% a 90 FPS. En total tiene grabado 55 horas de vídeo, compuestas por 11.5 millones de imágenes.

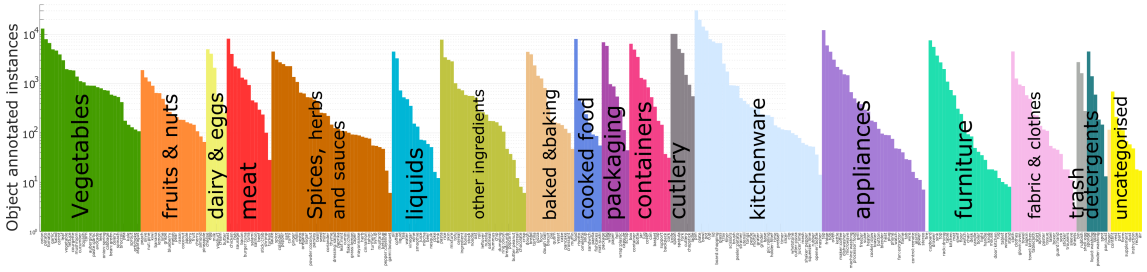


Figura 3.3: Figura extraída de (Damen y cols., 2020) donde se muestra el número de instancias de objetos anotadas

Por otra parte, EPIC-KITCHENS tiene etiquetados 39.594 segmentos de vídeo con los verbos de las acciones correspondientes a esos fragmentos de vídeo. Ver Figura 3.4.

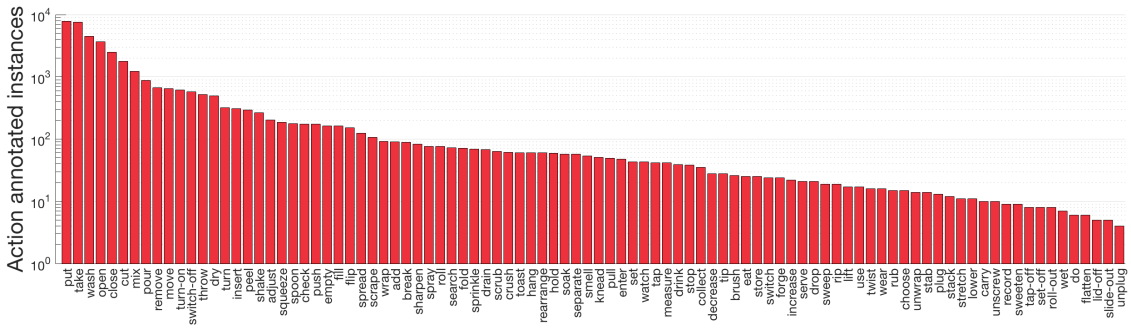


Figura 3.4: Figura extraída de (Damen y cols., 2020) donde se muestra el número de instancias de acciones anotadas

3.2 Análisis de los datos

Para empezar el análisis de los datos, nos fijamos en el repositorio donde se han subido las anotaciones[1] y el repositorio donde se han subido los scripts[2] para obtener las imágenes del dataset.

Del enlace de las anotaciones se ha descargado el archivo csv EPIC_train_object_labels.csv. Del enlace de los scripts de las anotaciones se ha clonado el repositorio, pero se ha lanzado solamente el script bash download_train.sh puesto que al no tener las anotaciones del conjunto de test, no podríamos calcular ninguna métrica de precisión sobre este dato.

Para realizar el análisis y procesado del fichero CSV he utilizado el paquete Pandas. Todo el procesado se ha realizado dentro de un jupyter notebook.

El formato que sigue cada línea del fichero CSV está dividido en 6 columnas. Ver Tabla 3.1.

Nombre columna	**Tipo**	**Ejemplo**	**Descripción**
noun_class	int	20	id de la clase
noun	string	bag	nombre original de la clase
id del participante	string	P01	id del participante
frame	int	056581	número del frame
bounding box	lista de 4-tuplas	"[(76, 1260, 462, 186)]"	formato: (<top:int>,<left:int>,<height:int>,<width:int>)

Tabla 3.1: Formato EPIC_train_object_labels.csv

Este CSV tiene en bruto 389.811 líneas. Deberemos procesar las líneas para eliminar nulos. Para realizar esto se ha usado Código 3.1. El resultado es un dataframe de 299.546 líneas.

Código 3.1: Lectura y procesamiento inicial de EPIC_train_object_labels.csv

```
df = pd.read_csv("/home/jesus/dataset/yolo_training/EPIC_train_object_labels.csv")
df.head() # Arroja como salidalas primeras 5 líneas del CSV
len(df) # Salida: 389811
df = df[df['bounding_boxes'].apply(lambda x: x) != '[]'] # Eliminas las líneas donde no hayan bounding boxes
len(df) # Salida: 299546
```

También es interesante la comprobación de si existe varias veces el mismo id del vídeo y el mismo frame en diferentes líneas. Ver Código 3.2. Esto se debe a que pueden ser diferentes bounding boxes

[1] `https://github.com/epic-kitchens/annotations` Accedido el 02/06/2020
[2] `https://github.com/epic-kitchens/download-scripts` Accedido el 02/06/2020

del mismo vídeo. Si fuese la misma clase dentro del mismo frame, se añadiría a la lista de bounding boxes de esa línea, o sea, si es una clase distinta, hay un duplicado de ese frame.

Código 3.2: Comprobación de duplicados

```
duplicated = df[df.duplicated(['video_id', 'frame'])]
len(duplicated) # 99156
```

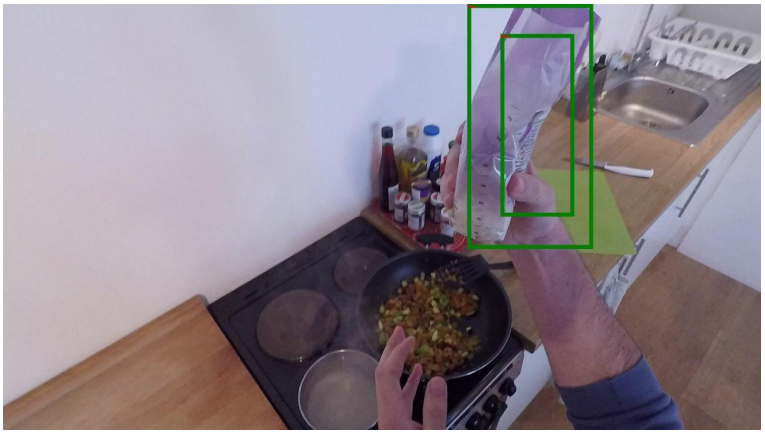
Para el número de clases, debemos tener en cuenta que la columna que marca valor que representa la clase en el CSV es el noun_class, no la columna noun. La columna noun es uno de los diferentes nombres que puede recibir esa clase. Ver Código 3.3.

Código 3.3: Número de clases en EPIC_train_object_labels.csv

```
print(len(df['noun'].unique())) # 784
print("Number of classes: {}".format(len(df['noun_class'].unique()))) # Number of classes: 290
```

Un ejemplo de imagen etiquetada en la que se han aplicado las anotaciones. Ver Figura 3.5.

(a) Imagen anotada del EPIC KITCHENS



(b) Imagen anotada del EPIC KITCHENS

Figura 3.5: Imágenes originales junto a sus bounding boxes

3.3 Procesamiento de los datos

El procesamiento de los datos es necesario para crear los ficheros train.txt y classes.txt que son requeridos en (qqwweee, 2018) y (david8862, 2020) que son las implementaciones de YOLO que se han usado. En el fichero train.txt se deberá escribir cada línea de la siguiente manera: ruta_imagen.jpg

x_min_0,y_min_0,x_max_0,y_max_0,$class\_id_0$ x_min_1,y_min_1,x_max_1,y_max_1,$class\_id_1$ ... box_n

Después, en el fichero classes.txt hemos de guardar todas las clases que queramos utilizar del dataset. Como se ha comentado, en el CSV se le asignan diferentes nombres a los objetos, pero solamente un número de clase. Por tanto, se han de agrupar los diferentes nombres en cada línea de classes.txt según el número de la clase. Por ejemplo, el número de la clase de 'apples' y 'apple' es el mismo, entonces se escribirá en el fichero apples_apple en el número de línea que le corresponda según el id de la clase.

Entonces, para la creación del fichero train.txt, se han renombrado todas las imagenes. Al descargar el dataset está organizado por carpetas, cada carpeta está nombrada según el id de un participante (p.e. P05) y las subcarpetas están nombradas según el formato IDParticipante_IDVideo. Cada imagen tiene está nombrada según el número de frame del video del que ha sido recogida, por ejemplo, 0000000031.jpg.

Por tanto, se han eliminado los ceros a la izquierda del número de frame, y se ha incorporado la etiqueta de la subcarpeta a la que perteneciese en el nombre de la imagen. Como resultado el formato del nombre será IDParticipante_IDVideo-nFrame.jpg. Ver Código 3.4 y Código 3.5.

Código 3.4: Procesamiento de los nombres de las imagénes

```
people = os.listdir(path)
for person in people:
  person_path = os.path.join(path, person)
  print(os.listdir(person_path)) #Lista de todos los videos de esa persona
  videos = os.listdir(person_path)
  for video in videos:
    video_path = os.path.join(person_path, video)
    print(os.listdir(video_path)) #Lista de todas los frames de los videos de persona x
    images = os.listdir(video_path)
    for image in images:
      image_path = os.path.join(video_path, image)
      print(image_path)
      shutil.copy(image_path,
                  '/content/gdrive/My Drive/Colab Notebooks/TFG/yolo_training/img/{}-{}'.format(video, image))
```

Código 3.5: Elimina los 0's a la izquierda del número de frame

```
imagenes = os.listdir(images_path)
for imagen in imagenes:
  imagen_nombre = imagen.split('-')
  imagen_nombre_final = '{}-{}'.format(imagen_nombre[0], imagen_nombre[1].lstrip('0'))
  os.rename(imagen, imagen_nombre_final)
  print(imagen_nombre_final)
```

Hecho esto, el siguiente paso fue crear un diccionario con clave el id de la clase y como valor una lista de nombres para ese id. Ejemplo: {97: ['napkin', 'napkins'], 234: ['apples', 'apple'], 186: ['kiwi', 'kiwis']}. Ver Código 3.6.

Código 3.6: Creación de diccionario id-nombres

```

diccionario_classes = {}
for index, row in df.iterrows():

    id_video = row['video_id']
    nFrame = row['frame']
    nClase = row['noun_class']
    nNameClass = row['noun']
    diccionario_classes.setdefault(nClase, [])
    if not nNameClass in diccionario_classes[nClase]:
        diccionario_classes[nClase].append('{}'.format(nNameClass))
```

Se remapea el número de clase puesto que faltan ids de clase en el CSV que hemos usado, y también porque posteriormente se deseará la creación de un subconjunto de clases. Ver Código 3.7. Por ejemplo, la clase 20 ahora será la 0, la 28 la 1, la 40 la 2, etc.

Código 3.7: Remapeo de id de clases

```
1
2 dict_num_nounclass = {}
3 dict_nounclass_num = {}
4 diccionario_classes_nuevo = {}
5 for i, item in enumerate(diccionario_classes.keys()):
6     dict_nounclass_num[item] = i
7     dict_num_nounclass[i] = item
8     print("{}={}".format(i, item))
9     diccionario_classes_nuevo[i] = diccionario_classes[item]
```

Si recorremos el nuevo diccionario de manera ordenada, podemos escribir ya el fichero classes.txt. Ver Código 3.8.

Código 3.8: Escribir classes.txt

```
1
2 with open('/home/jesus/dataset/yolo_training/classes.txt', 'w') as f:
3     for key in sorted (diccionario_classes_nuevo.keys()):
4         #cadena = "{}/".format(key)
5         cadena = ""
6         linea = diccionario_classes_nuevo.get(key)
7         for i, item in enumerate(linea):
8             item = item.replace(" ", "-")
9             if i+1 != len(linea):
10                cadena += '{}_'.format(item)
11            else:
12                cadena += '{}'.format(item)
13        print(cadena)
14        f.write("%s\n" % cadena)
```

El resultado es un fichero como en Código 3.9.

Código 3.9: classes.txt

```
1 bag_rice-bag_cereal-bag_flour-bag_mozzarella-bag_cereal-bags_onion-bag_pasta-bag_bags_bin-bag_sandwich↩
      ↪ -bag_plastic-bag_plastic-bags_kitchen-bag_trash-bag_cheese-bag_spaghetti-bag_grapes-bag_oregano-↩
      ↪ bag_coffee-bag
2 bottle_oil-bottle_sauce-bottle_milk-bottle_pesto-bottle_vinegar-bottles_bottles_water-bottle
3 carrot_carrots_tail-carrot
4 fork_forks
5 glass_glasses
6 knife_blade_knives_mezzaluna_mincing-knife_mezzaluna-knife_chopper_moon-chopper
7 lid_yoghurt-lid_pot-lid_pan-lid_chopper-lid_teapot-lid_box-lid_saucepan-lid_bottle-lid_tin-lid
8 pan_sauce-pan_saucepan_frying-pan_pans_wok_content-pan_cake-pan_sauce-pans
9 microwave_microwave-oven
10 egg-shell_eggshell_shell_eggshells_shells
11 napkin_napkins
12 apples_apple
13 kiwi_kiwis
```

Entonces, para la escritura del fichero train.txt se lanza el Código 3.10. Donde se remapea la clase que se lea del dataframe a su nuevo id. Cabe añadir, que se procesa también la cadena de las bounding boxes. En este código se emplea un diccionario donde las claves son las rutas de la imágenes y los valores una lista de bounding boxes. Las bounding boxes se modifican puesto que el formato que nos piden en (david8862, 2020) es diferente al que nos dan en las anotaciones, esto se puede ver en la línea 29.

Código 3.10: Creación de diccionario para almacenar rutas y bounding boxes

```
1
2 diccionario = {}
```

```
3
4for index, row in df.iterrows():
5    id_video = row['video_id']
6    nFrame = row['frame']
7    nClase = row['noun_class']
8    stringImagen = '{}-{}.jpg'.format(str(id_video), str(nFrame))
9    path_image = os.path.join(path, stringImagen)
10   box = row['bounding_boxes'].replace("[", "")
11   box = box.replace("]", "")
12   box = box.replace("(", "")
13   box = box.replace(")", "|")
14   box = box.replace("|,", "/")
15   box = box.replace("|", "")
16   box = box.replace(", ", ",")
17   boxes = box.split("/")
18   print(boxes)
19   for i in range(len(boxes)):
20       boxes[i] = boxes[i].replace(" ", "")
21   for i, item in enumerate(boxes):
22       print(item)
23       box = boxes[i].split(',')
24       print(box)
25       print("top: {}".format(box[0]))
26       #print(type(int(box[0])))
27       print("left: {}".format(box[1]))
28       nuevo_string = "{},{},{},{}".format(box[1], box[0], str(int(box[1])+int(box[3])), str(int(box[0])+int(box[2])))
29       print(nuevo_string)
30       print(path_image)
31       diccionario.setdefault(path_image, [])
32       print("nClase antes: {}".format(nClase))
33       clase = dict_nounclass_num[nClase]
34       #clase = nClase
35       print("nClase despues: {}".format(clase))
36       diccionario[path_image].append('{},{}'.format(nuevo_string, clase))
```

Lanzamos entonces el código para escribir un fichero. Ver Código 3.12

Código 3.11: Escribir train.txt

```
1
2with open('/home/jesus/dataset/yolo_training/train.txt', 'w') as f:
3    for key in diccionario.keys():
4        cadena = ""
5        cadena = "{} ".format(key)
6        linea = diccionario.get(key)
7        for i, item in enumerate(linea):
8            if i != len(linea)-1:
9                cadena += '{} '.format(item)
10           else:
11               cadena += '{}'.format(item)
12       f.write("%s\n" % cadena)
```

Código 3.12: classes.txt

```
1
2with open('/home/jesus/dataset/yolo_training/train.txt', 'w') as f:
3    for key in diccionario.keys():
4        cadena = ""
5        cadena = "{} ".format(key)
6        linea = diccionario.get(key)
7        for i, item in enumerate(linea):
8            if i != len(linea)-1:
9                cadena += '{} '.format(item)
10           else:
11               cadena += '{}'.format(item)
12       f.write("%s\n" % cadena)
```

El resultado es un fichero con 46015 líneas, una por cada línea del diccionario. Ver Código 3.13.

Código 3.13: train.txt

```
/media/compartido/TFG/images/P01\_01-56581.jpg 1260,76,1446,538,0 1176,0,1494,620,0
media/compartido/TFG/images/P01_01-56971.jpg 954,652,1238,874,0 872,708,1220,910,0
```

3.4 Creación subconjunto de datos

La creación del subconjunto de datos viene dado por el hecho de que no queríamos entrenar con todas las imágenes para que la tarea de entrenamiento no fuese demasiado prolongada y que fuesen unos objetos representativos para cada tipo de agarre que pudiera darse. Por ello se han seleccionado las clases que se han puesto en Código 3.9. No nos basamos en la cantidad de imágenes de cada clase.

Para la creación de este subset se han realizado los mismos pasos que previamente, solamente filtrando por el id de las clases deseadas. Se modifican Código 3.6 y Código 3.10. Los códigos resultantes son Código 3.14 y 3.15.

Código 3.14: Creación de diccionario id-nombres reducido

```
diccionario_classes = {}
lista_reducida = [18, 16, 11, 5, 1, 28, 11, 20, 186, 40, 234, 97, 249, 55]

for index, row in df.iterrows():

    id_video = row['video_id']
    nFrame = row['frame']
    nClase = row['noun_class']
    nNameClass = row['noun']
    diccionario_classes.setdefault(nClase, [])
    if nClase in lista_reducida and not nNameClass in diccionario_classes[nClase]:
        diccionario_classes[nClase].append('{}'.format(nNameClass))
```

Código 3.15: Creación de diccionario reducido para almacenar rutas y bounding boxes

```
diccionario = {}
lista_reducida = [18, 16, 11, 5, 1, 28, 11, 20, 186, 40, 234, 97, 249, 55]
for index, row in df.iterrows():
    id_video = row['video_id']
    nFrame = row['frame']
    nClase = row['noun_class']
    stringImagen = '{}-{}.jpg'.format(str(id_video), str(nFrame))
    path_image = os.path.join(path, stringImagen)
    if nClase in lista_reducida:
        box = row['bounding_boxes'].replace("[", "")
        box = box.replace("]", "")
        box = box.replace("(", "")
        box = box.replace(")", "|")
        box = box.replace("|,", "/")
        box = box.replace("|", "")
        box = box.replace(", ", ",")
        boxes = box.split("/")
        print(boxes)
        for i in range(len(boxes)):
            boxes[i] = boxes[i].replace(" ", "")
        for i, item in enumerate(boxes):
            print(item)
            box = boxes[i].split(',')
            print(box)
            print("top: {}".format(box[0]))
            #print(type(int(box[0])))
            print("left: {}".format(box[1]))
            nuevo_string = "{},{},{},{}".format(box[1], box[0], str(int(box[1])+int(box[3])), str(int(box[0])+int(box[2]))
            print(nuevo_string)
            print(path_image)
            diccionario.setdefault(path_image, [])
            print("nClase antes: {}".format(nClase))
```

```
            clase = dict_nounclass_num[nClase]
            #clase = nClase
            print("nClase despues: {}".format(clase))
            diccionario[path_image].append('{},{}'.format(nuevo_string, clase))
```

3.4.1 Análisis del subset

Para el análisis del subset, listamos mediante el Código 3.16.

Código 3.16: Código para generar gráfica que muestra por cada número de clase cuantas ocurrencias hay

```
import matplotlib.pylab as plt
lista_reducida = [18, 16, 11, 5, 1, 28, 11, 20, 186, 40, 234, 97, 249, 55]
dictionary = {}
for i in lista_reducida:
    dictionary.setdefault(i, [])
    dictionary[i] = len(df[df['noun_class'].apply(lambda x: x) == i])
    print(len(df[df['noun_class'].apply(lambda x: x) == i]))
print(dictionary)

plt.bar(range(len(dictionary)),dictionary.values(),tick_label=dictionary.keys())
plt.savefig('bar.png')
plt.show()
```

La gráfica resultante es la que se puede ver en Figura 3.6 y la tabla resumen es Tabla 3.2.

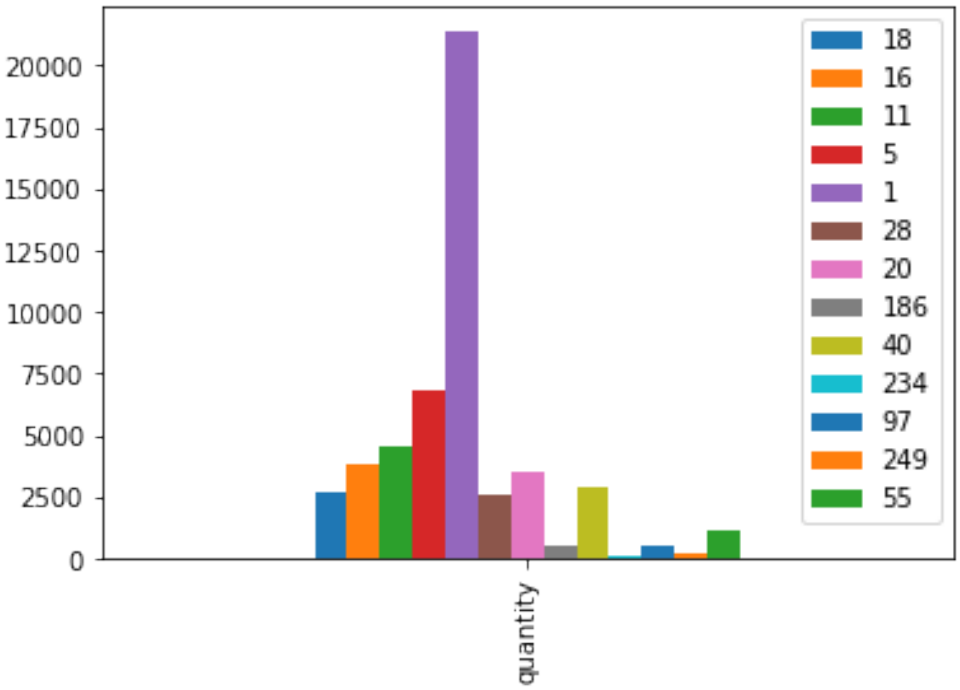


Figura 3.6: Gráfica de número de ocurrencias por clase

Número clase	Nombre clase	cantidad
18	fork	2674
16	glass	3818
11	lid	4500
5	knife	6796
1	pan	21385
28	bottle	2561
20	bag	3517
186	kiwi	496
40	carrot	2904
234	apple	138
97	napkin	514
249	eggshell	214
55	microwave	1156

Tabla 3.2: Clases y cantidad de instancias por clase

4 Experimentación

Durante este capitulo se tratará los experimentos, pruebas, evaluación y análisis sobre los entrenamientos de las arquitecturas de redes neuronales probadas. Explicando primero qué configuraciones se han usado para el entrenamiento, después se explicará cuánto han durado los entrenamientos y qué resultados se han obtenido según la arquitectura entrenada.

4.1 Entrenamiento

4.1.1 Descarga de modelos

Para la obtención de los modelos y posteriormente convertirlos en modelos usables por Keras debemos lanzar los comandos de Código 4.1 en una consola, dependiendo del modelo que queramos descargar y convertir.

Código 4.1: Comandos para la descarga y conversión de los modelos

```
1
2 wget -O weights/darknet53.conv.74.weights https://pjreddie.com/media/files/darknet53.conv.74
3 python tools/convert.py cfg/darknet53.cfg weights/darknet53.conv.74.weights weights/darknet53.h5
4 wget -O weights/yolov4.weights https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/↩
      ↪ yolov4.weights
5 python tools/convert.py --yolo4_reorder cfg/yolov4.cfg weights/yolov4.weights weights/yolov4.h5
6 wget -O weights/yolov3.weights https://pjreddie.com/media/files/yolov3.weights
7 python tools/convert.py cfg/yolov3.cfg weights/yolov3.weights weights/yolov3.h5
```

4.1.2 Configuración del entrenamiento

Para configurar el entrenamiento, debemos modificar el fichero train.py que se encuentra en el repositorio de (david8862, 2020). Los argumentos que han sido modificados a lo largo de los diferentes entrenamientos son:

- Opciones del modelo.
- Opciones de los datos.
- Opciones de entrenamiento.
- Opciones de evaluación.

Para las opciones del modelo existen los argumentos:

- –model_type: El modelo de YOLO que queramos usar.
- –anchors_path: Ruta a los anchors de YOLO.
- –model_image_size: Resolución de la imagen de input.
- –weigths_path: Pesos/modelo preentrenado para realizar fine-tuning.

Para el tipo de modelo se ha probado con yolo3_darknet, yolo4_darknet, yolo3_efficientnet. En el caso del anchors_path ha sido siempre el mismo, yolo3_anchors.txt. La resolución de las imágenes ha sido siempre 416x416. Y el argumento de los pesos ha sido utilizada la ruta de los pesos/modelo de Darknet53 y también la ruta de los pesos en los casos que el entrenamiento haya tenido que detenerse por diferentes motivos. Por el hecho de la resolución de las imágenes se vuelve a dimensionar la imagen, añadiendo unos bordes grises, se vuelven a situar las bounding boxes también. Ver Figura 4.1.

Figura 4.1: Procesamiento de la imagen para el entrenamiento

Para las opciones de los datos existen los argumentos:

- –annotation_file: Ruta al fichero donde se encuentran las anotaciones. En nuestro caso es el fichero train.txt.
- –val_annotation_file: Ruta al fichero donde se encuentran las anotaciones del conjunto de validación, si es nulo se coge del annotation_file.
- –val_split: Es el porcentaje del conjunto de entrenamiento que se usará como conjunto de validación.
- –classes_path: Ruta al fichero donde se encuentran las clases. En nuestro caso es el fichero classes.txt.

En el caso de –annotation_file se ha designado la ruta al train.txt. Por otra parte, para –val_annotation_file lo hemos mantenido en nulo, puesto que queremos dividir train.txt en un conjunto de validación y otro de entrenamiento. Para el –val_split se ha establecido en un 20% para todos los entrenamientos, posteriormente hablaremos de cómo se ha creado ese conjunto de validación. Para el argumento –classes_path se ha establecido la ruta al fichero classes.txt.

Para las opciones de entrenamiento existen los argumentos:

- –batch_size: Las imágenes que se le pasarán al red en cada paso de las épocas del entrenamiento.

- –optimizer: El optimizador que se utilizará.
- –learning_rate: El ratio de aprendizaje que se usará.
- –decay_type: La función por la que decaerá el ratio de aprendizaje en el entrenamiento.
- –transfer_epoch: Las épocas en las que consista la fase de transfer learning.
- –freeze_level: Define qué capas están congeladas en la red neuronal
- –init_epoch: La epoca en la que se empieza el fine-tuning, por defecto está a 0.
- –total_epoch: La cantidad de épocas en total, por defecto a 250.
- –multiscale: Si queremos o no entrenamiento a multiescala.
- –rescale_interval: Número de intervalo de iteraciones para reescalar las dimensiones de la entrada.
- –model_pruning: Si queremos realizar model pruning para optimizar el entrenamiento.
- –label_smoothing: Si queremos realizar suavizado de etiquetas para la función de perdida de la clasificación
- –data_shuffle: Si queremos barajar las líneas de entrenamiento y validación para realizar validación cruzada.
- –gpu_num: El número de GPUs que se utilizarán durante el entrenamiento.

Para –batch_size hemos usado 8 para el entrenamiento de la yolov3_darknet y 4 para el resto, esto se realiza para no sobrepasar la memoria de la GPU, a más memoria, más grande puede ser el tamaño del batch. Para –optimizer hemos usado el optimizador adam. En el caso de –learning_rate se ha fijado a 0.001 en un principio. El argumento –decay_type se ha dejado en nulo. El argumento –transfer_epoch se ha establecido en 20 épocas para transfer learning. El argumento –freeze_level varía, para el primer entrenamiento con yolov3_darknet se ha usado toda la red descongelada, para los otros casos se ha congelado el backbone y se ha entrenado el resto, posteriormente hablaremos de cuántos parámetros supone esto. Para –init_epoch y –total_epoch se ha dejado la opción por defecto. Para –multiscale se ha marcado que no se use. El intervalo de –rescale_interval es de 10. Para –model_pruning, –label_smoothing y –data_shuffle se ha seleccionado la opción de no ser usados. El número de GPUs –gpu_num se ha establecido a 1.

Para las opciones de evaluación existen los argumentos:

- –eval_online: Si queremos realizar la evaluación sobre el conjunto de validación durante el entrenamiento.
- –eval_epoch_interval: El intervalo entre épocas para lanzar la evaluación durante el entrenamiento.
- –save_eval_checkpoint: Si queremos guardar como checkpoint los pesos/modelo con el mejor resultado en la evaluación.

Se ha marcado que se realice la evaluación durante el entrenamiento, durante los primeros entrenamientos se usó que para cada época se realizase una evaluación, según se ha comprobado empíricamente esto no aportaba ningún valor, por tanto, se estableció posteriormente a 3. Por otra parte, también guardamos el resultado que arroje mejor promedio de precisión media.

Para la creación del conjunto de validación y de entrenamiento lo que se realiza es lo que aparece en Código 4.2. Barajamos las líneas de las anotaciones, y marcamos una semilla aleatoria. Esta semilla

siempre será la misma, puesto que se quiere que cuando volvamos a lanzar nuevamente el entrenamiento vuelva ser el mismo conjunto de validación. Sabemos multiplicando la longitud del dataset por el porcentaje que queramos que sea nuestro conjunto de validación.

Código 4.2: Código para generar conjuntos de entrenamiento y de validacion

```
1
2 dataset = get_dataset(annotation_file)
3 val_split = args.val_split
4 num_val = int(len(dataset)*val_split)
5 num_train = len(dataset) - num_val
6
7 def get_dataset(annotation_file, shuffle=True):
8     with open(annotation_file) as f:
9         lines = f.readlines()
10        lines = [line.strip() for line in lines]
11
12    if shuffle:
13        np.random.seed(10101)
14        np.random.shuffle(lines)
15        np.random.seed(None)
16
17    return lines
```

Para obtener las anchors y las clases hacemos lo que se muestra en Código 4.3

Código 4.3: Código para obtener las clases y las anchors

```
1
2 class_names = get_classes(classes_path)
3 num_classes = len(class_names)
4
5 anchors = get_anchors(args.anchors_path)
6 num_anchors = len(anchors)
7
8 def get_classes(classes_path):
9     '''loads the classes'''
10    with open(classes_path) as f:
11        class_names = f.readlines()
12    class_names = [c.strip() for c in class_names]
13    return class_names
14
15 def get_anchors(anchors_path):
16    '''loads the anchors from a file'''
17    with open(anchors_path) as f:
18        anchors = f.readline()
19    anchors = [float(x) for x in anchors.split(',')]
20    return np.array(anchors).reshape(-1, 2)
```

Los callbacks usados son los que se pueden ver en 4.4. Donde usamos un callback para registrar el entrenamiento en la Tensorboard provista por Tensorflow. Otro callback usado para registrar los puntos de guardado, donde guardamos todo el modelo y solamente el que mejor val_loss tenga, esta comprobación se realiza en cada época del entrenamiento. Los callback ReduceLROnPlateau y EarlyStopping sirven para que si la métrica val_loss se estanca se reduzca el learning rate, y si no mejora después de N épocas se para el entrenamiento, pues se supone que no puede mejorar más. Más adelante en este documento se explicará el callback para la evaluación.

Código 4.4: Código donde realizan las llamadas a los callbacks

```
1
2 logging = TensorBoard(log_dir=log_dir, histogram_freq=0, write_graph=False, write_grads=False, write_images=False
      ↪ , update_freq='batch')
3 checkpoint = ModelCheckpoint(os.path.join(log_dir, 'ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'),
4     monitor='val_loss',
5     verbose=1,
6     save_weights_only=False,
7     save_best_only=True,
8     period=1)
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=1, cooldown=0, min_lr=1e-10)
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1)
terminate_on_nan = TerminateOnNaN()
eval_callback = EvalCallBack(args.model_type, dataset[num_train:], anchors, class_names, args.model_image_size, args.model_pruning, log_dir, eval_epoch_interval=args.eval_epoch_interval, save_eval_checkpoint=args.save_eval_checkpoint)
    callbacks.append(eval_callback)
```

Para el entrenamiento se ha utilizado una GPU NVIDIA GeForce GTX 1080.

4.2 Evaluación sobre datos de validación

Para la evaluación de los datos nos fijaremos en el callback de evaluación, por qué imágenes está compuesto el conjunto de validación y con qué métrica los evaluamos.

Como vemos en Código 4.5 se llama al método eval_AP al final de cada época, y según lo establecido antes, si el mAP se mejora, nos guardamos ese modelo. Usamos un threshold del 0.5 para el IoU (Intersection over Union), la intersección entre la bounding box detectada y la del ground truth, también usamos un threshold del 0.1 para la confianza, así descartamos detecciones con 0% de confianza. Empíricamente se ha probado que un threshold más bajo en ambos provoca arrojar un mAP más alto.

Código 4.5: Método del callback de evaluación donde se calcula el mAP

```

def on_epoch_end(self, epoch, logs=None):
    if (epoch+1) % self.eval_epoch_interval == 0:
        # Do eval every eval_epoch_interval epochs
        eval_model = self.update_eval_model(self.model)
        mAP = eval_AP(eval_model, 'H5', self.annotation_lines, self.anchors, self.class_names, self.model_image_size, eval_type='VOC', iou_threshold=0.5, conf_threshold=0.1, save_result=False)
        if self.save_eval_checkpoint and mAP > self.best_mAP:
            # Save best mAP value and model checkpoint
            self.best_mAP = mAP
            self.model.save(os.path.join(self.log_dir, 'ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}-mAP{mAP:.3f}.h5'.format(epoch=(epoch+1), loss=logs.get('loss'), val_loss=logs.get('val_loss'), mAP=mAP)))
```

Para explicar la métrica mAP[1], debemos entender previamente conceptos como precisión y recall. La precisión es cómo de preciso son las predicciones de nuestro modelo (el porcentaje de las predicciones correctas). El recall es cómo de bien se detectan los positivos. Ver Figura 4.2.

$$Precision = \frac{TP}{TP + FP} \qquad TP = \text{True positive}$$

$$TN = \text{True negative}$$

$$Recall = \frac{TP}{TP + FN} \qquad FP = \text{False positive}$$

$$FN = \text{False negative}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Figura 4.2: Fórmulas precisión y recall

Decimos que una detección es correcta cuando su IoU es mayor a nuestro threshold. Ver Figura 4.3.

[1] https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173 Accedido el 02/06/2020

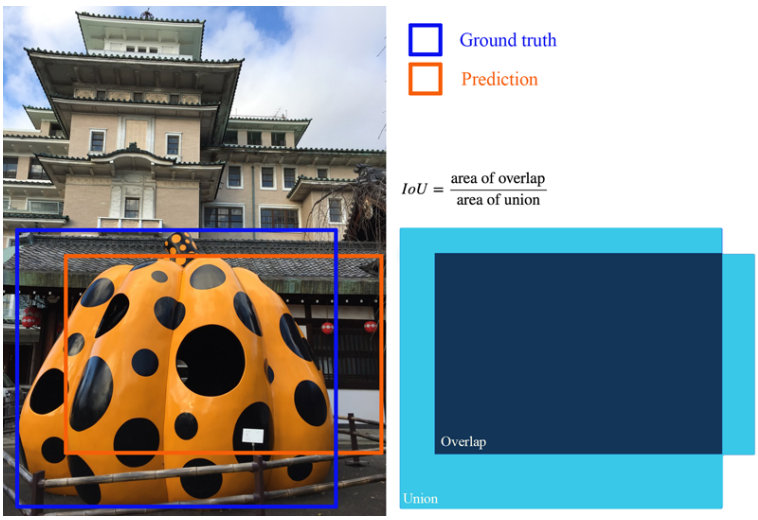


Figura 4.3: Fórmula y demostración Intersection over Union

Para el calculo de la AP (average precision) se calcula el area bajo la curva precision-recall (Ver Figura 4.4). Debemos eliminar el zigzagueo previamente para poder calcular el área bajo la curva (Ver Figura 4.5), la curva resultante es la de verde. La precisión media se puede calcular de dos maneras, en el desafío PASCAL VOC2008 se usó una precisión media interpolada por 11 puntos (ver Figura 4.6). Por otra parte, existe calcular el área de los rectángulos que se forman cuando el valor máximo de la precisión decrece (Ver Figura 4.7), esto sería mediante AUC (Area Under Curve).

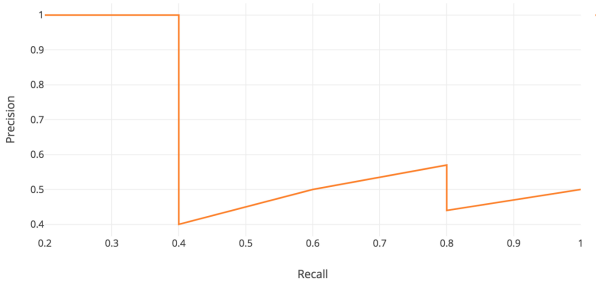


Figura 4.4: Curva precision-recall

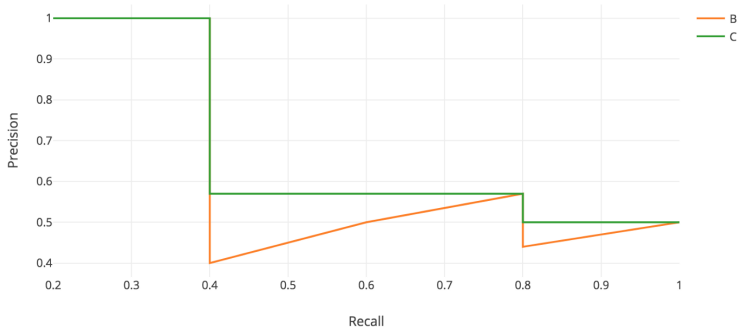
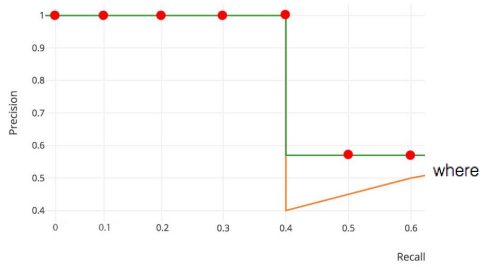
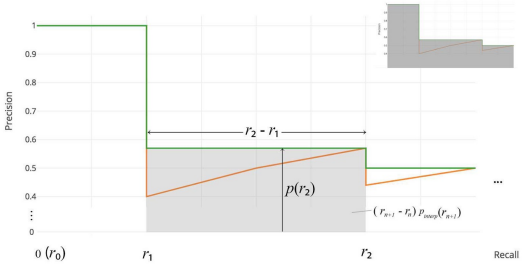


Figura 4.5: Curva precision-recall sin zigzagueo



$$AP = \frac{1}{11} \sum_{r \in \{0.0,\ldots,1.0\}} AP_r$$
$$= \frac{1}{11} \sum_{r \in \{0.0,\ldots,1.0\}} p_{interp}(r)$$
$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r})$$

Figura 4.6: Precisión media interpolada



$$AP = \sum (r_{n+1} - r_n)\, p_{interp}(r_{n+1})$$
$$p_{interp}(r_{n+1}) = \max_{\tilde{r} \geq r_{n+1}} p(\tilde{r})$$

Figura 4.7: Precisión media calculada mediante AUC

Retomando la formación de nuestro conjunto de validación, podemos observar en Figura 4.8. Está compuesto por 13 clases y 9203 imágenes. Se puede observar en la figura cuántas anotaciones hay por clase en el conjunto de validación, vemos que hay muy pocas de clases como 'apple' esto puede causar problemas posteriormente, pero en el conjunto de entrenamiento había aproximadamente 150.

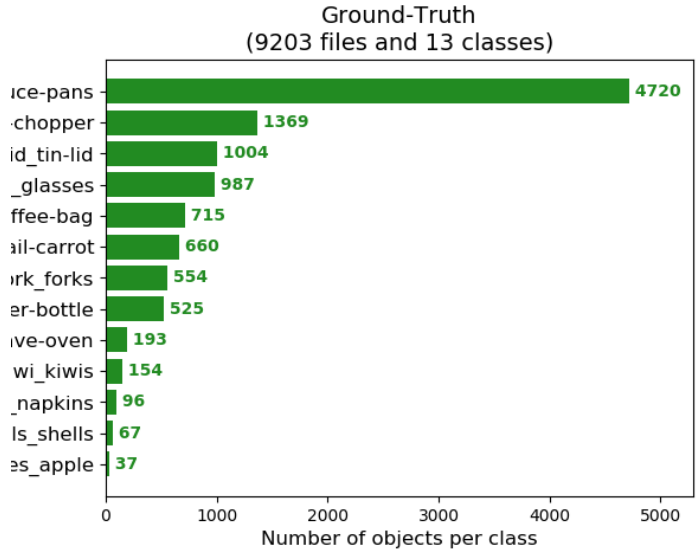
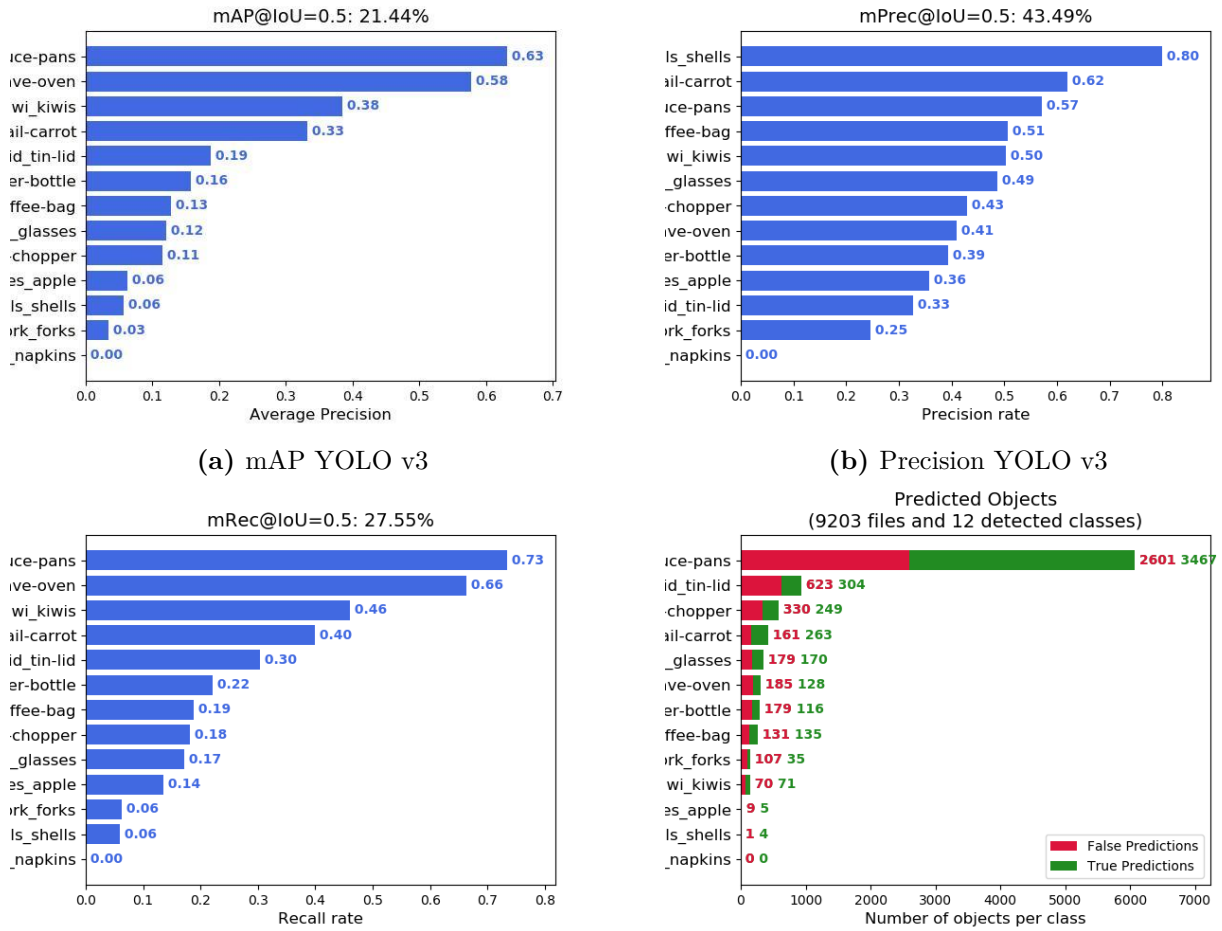


Figura 4.8: Ground Truth del validation set

4.2.1 Análisis de los resultados

Aquí se analizan los resultados obtenidos sobre la evaluación del conjunto de validación según las arquitecturas y configuraciones utilizadas. Todos estos resultados están analizados con un threshold para el IoU de 0.5 y 0.1 para el threshold de la confianza.

Para el entrenamiento de la YOLO v3 con backbone Darknet53 y todas las capas descongeladas, con un batch size de 8, se tomó 50 épocas y los resultados fueron los de Figura 4.9.



(a) mAP YOLO v3

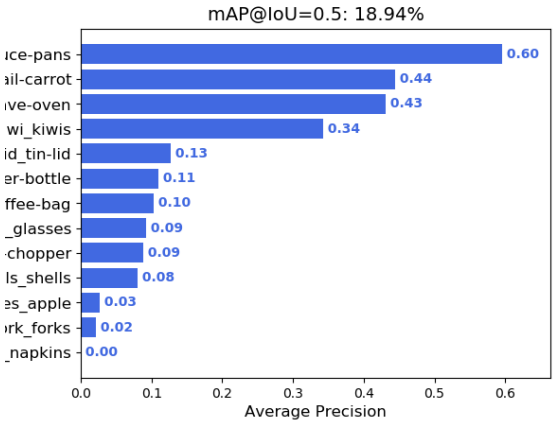
(b) Precision YOLO v3

(c) Recall YOLO v3

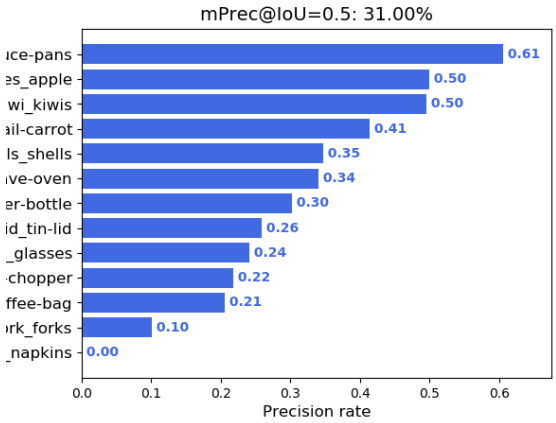
(d) Información de predicciones

Figura 4.9: Gráficas entrenamiento YOLO v3 + Darknet53

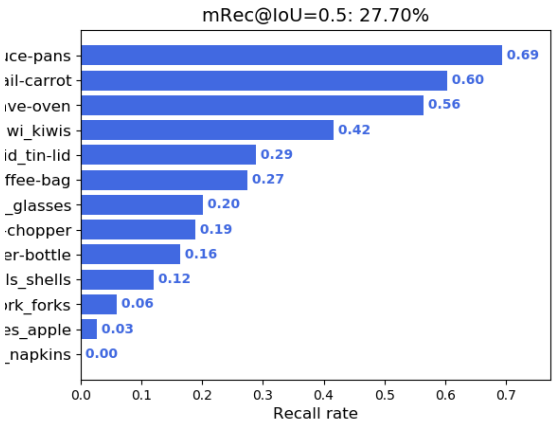
Para el entrenamiento de la YOLO v4 con backbone Darknet53 sin descongelar el backbone, con un batch size de 4, se tomó 40 épocas y los resultados fueron los siguientes de la Figura 4.10.



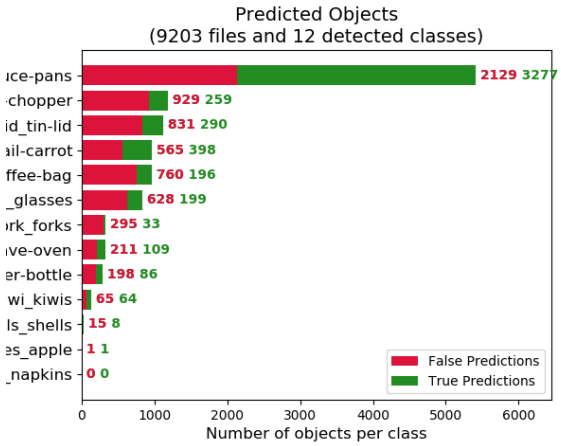
(a) mAP YOLO v4



(b) Precision YOLO v4



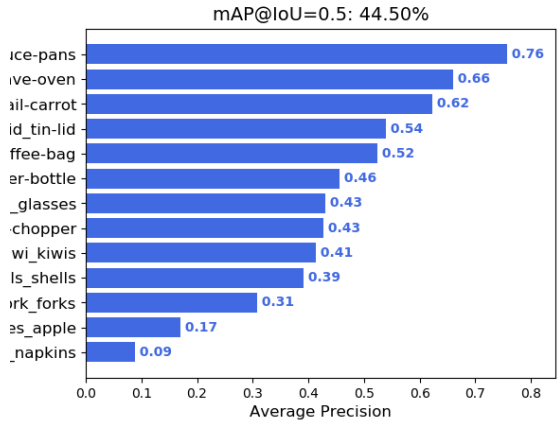
(c) Recall YOLO v4



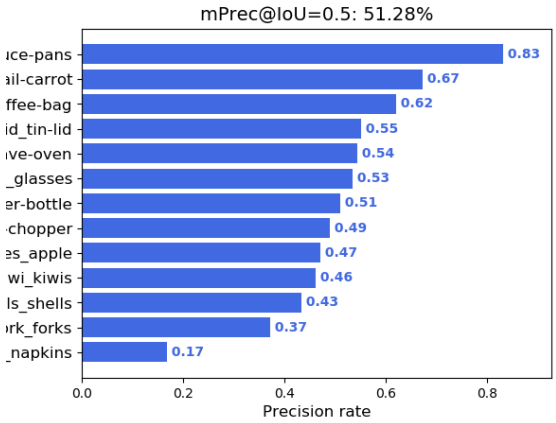
(d) Información de predicciones

Figura 4.10: Gráficas entrenamiento YOLO v4

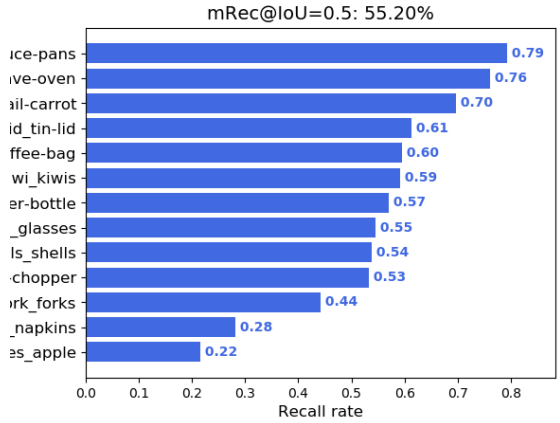
Para el entrenamiento de la YOLO v3 con backbone EfficientNet sin descongelar el backbone, con un batch size de 4, se tomó 45 épocas y los resultados fueron los de Figura 4.11.



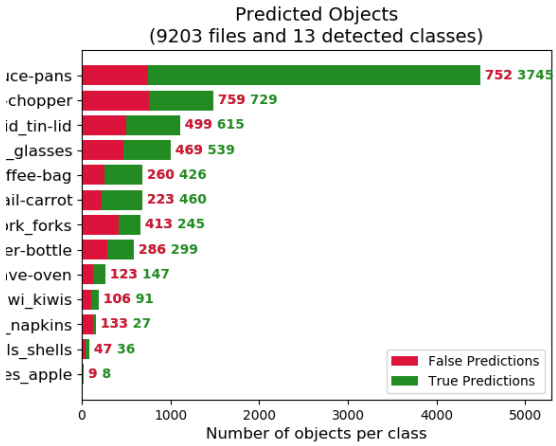
(a) mAP YOLO v3



(b) Precision YOLO v3



(c) Recall YOLO v3



(d) Información de predicciones

Figura 4.11: Gráficas entrenamiento YOLO v3 + EfficientNet

4.3 Comparativa entre arquitecturas

La comparativa se basa en el análisis de los resultados del punto anterior. Ver Tabla 4.1.

Nombre Arquitectura	mAP	Recall	Precisión	Épocas
YOLO v3 + Darknet53	21.44%	27.55%	43.49%	50
YOLO v4 + Darknet53	18.94%	27.70%	31.00%	40
YOLO v3 + EfficientNet	44.50%	55.20%	51.28%	45

Tabla 4.1: Comparativa de resultados entre arquitecturas

En el caso de YOLOv4 + Darknet53 y YOLOv3 + EfficientNet los parámetros que se podían entrenar fueron la mitad, puesto que se dejó sin descongelar las capas del backbone. En cambio, en el caso de YOLOv3 + Darknet53 la cantidad de parámetros a entrenar fueron 62M debido a que se descongeló la red entera.

En la comparativa de mAP con (Damen y cols., 2020) se comparará con los resultados de YOLOv3 + EfficientNet debido a que es la que mejor resultados nos ha dado. Bajo las mismas condiciones de

threshold para IoU los resultados para las clases comunes que son 'pan', 'knife, y 'lid' fueron:

En (Damen y cols., 2020) usando (Ren y cols., 2015) como modelo han sido, 67,60%, 28.80% y 58,48% respectivamente. Mientras que con nuestro modelo hemos obtenido 76%, 43% y 54% respectivamente. La comparativa solamente se puede realizar con esas clases y no el mAP global puesto que no evaluamos las mismas clases ni el mismo número de clases.

5 Resultados

5.1 Inferencia sobre imágenes

La Figura 5.1 muestra algunas imágenes donde se puede observar las detecciones realizadas. La etiqueta de color negro es la que fue etiquetada, las del resto de colores son las detecciones realizadas por el modelo YOLOv3 + EfficientNet.

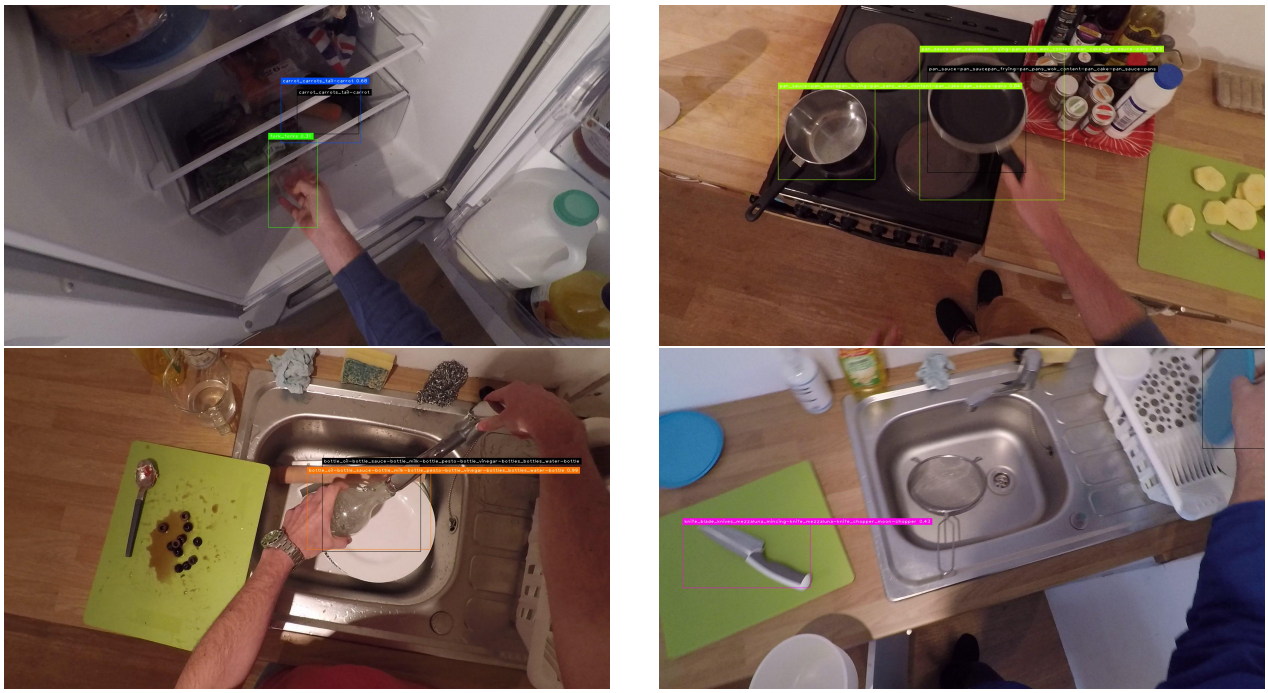


Figura 5.1: Imágenes detectadas por YOLOv3+EfficientNet

Para imágenes de una cocina fuera del dataset y con una resolución de imagen (que luego se vuelve dimensionar al pasarle la imagen a la red) los resultados son los de la Figura 5.2. Estas fotos se han realizado con la cámara de un móvil OnePlus 3T. El tiempo de inferencia por imagen es de menos de 0.1 segundo.

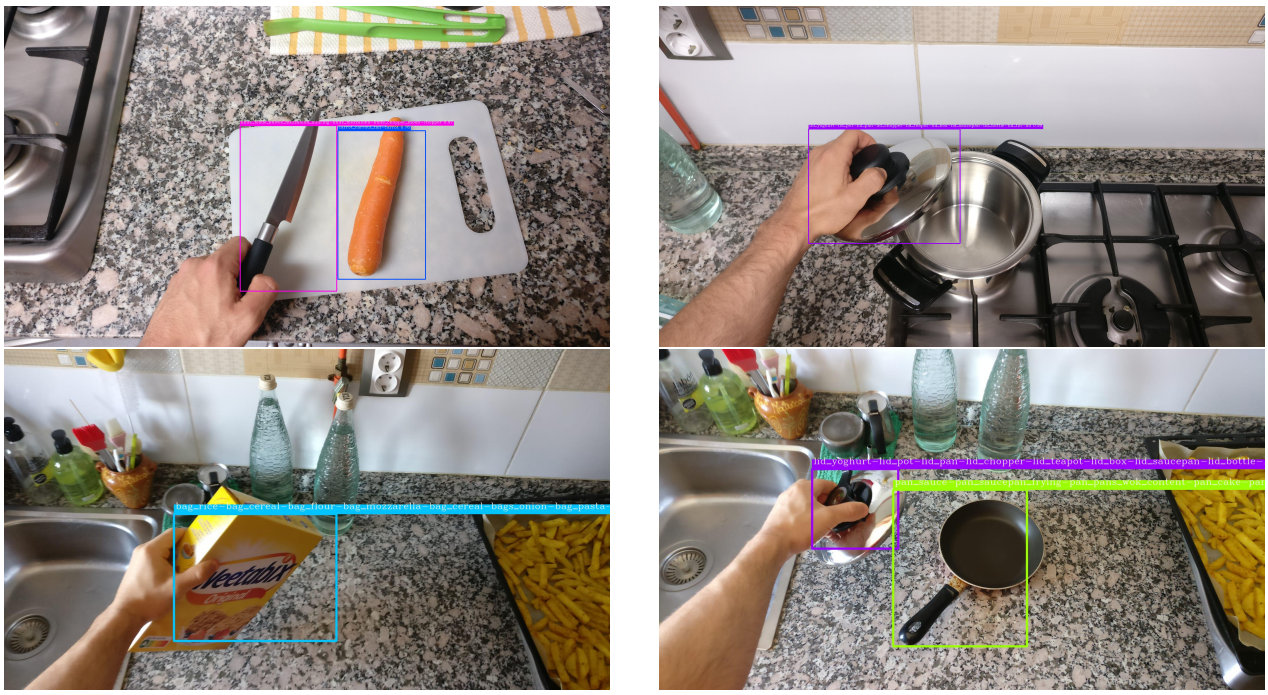


Figura 5.2: Imágenes detectadas por YOLOv3+EfficientNet en cocina real

Mediante el enlace `https://drive.google.com/file/d/1Epg4C-56xDbQtPMqIL5rNy6gGQlXWw_z/view` se puede descargar y ver un ejemplo de vídeo donde se detectan las clases de objetos entrenados.

6 Conclusiones

Para concluir el documento, se especifica algunos de los posibles trabajos futuros para mejorar el sistema que asistirá al exoguante.

6.1 Trabajo futuro

En primer lugar, se deberá revisar si es suficiente la precisión y velocidad de los resultados obtenidos y de los algoritmos entrenados. La mejora de la precisión se puede llevar a cabo mediante la ampliación del número de imágenes de alguna clase usando aumentado de datos, también se podría entrenar durante más épocas los algoritmos, incluso se podría seleccionar otras clases o conjunto de datos que incluyan más imágenes. Durante este proyecto no se utilizó el aumentado de datos, porque se pensó que al incluir diferentes tipos de objetos en la misma clase "madre" ya el conjunto sería lo suficientemente variado.

Por otra parte, el predecir la acción que realizará el observador o usuario dependiendo de los objetos activos podría servir también para la inferencia del tipo de agarre. Esto también se podría realizar usando el mismo conjunto de datos, gracias a que provee de vídeos con las acciones anotadas.

Además, el tipo de agarre se podría inferir por unas reglas establecidas, pero sería interesante explorar la opción del uso de algoritmos de aprendizaje por refuerzo, o algoritmos de aprendizaje profundo para realizar la inferencia sobre el tipo de agarre.

Los tipos de agarre que se han planteado como óptimos son los que se pueden observar en el manual de usuario de Ottobock[1], debido a que se ha observado diferentes alternativas en el mercado de las prótesis que implementan estos agarres o similares.
Otro objetivo futuro sería conseguir que el sistema de actuación adopte estos agarres mediante los servomotores que se emplean en el diseño.

Y por último, evidentemente, que al integrar estos nuevos sistemas de percepción e inferencia sobre el tipo de agarre con el sistema de actuación se efectúe de una manera amigable para el usuario, consiguiendo que no sea percibido como intrusivo. Es de gran relevancia el conseguir que el usuario se sienta lo más cómodo con estas nuevas ayudas, si no puede ser usado cómodamente, no se alcanzaría el objetivo perseguido, mejorar la calidad de vida del usuario.

[1] `https://www.ottobockus.com/prosthetics/upper-limb-prosthetics/solution-overview/bebionic-hand/` Accedido el 03/06/2020

Bibliografía

Alexey Bochkovskiy, H.-Y. M. L., Chien-Yao Wang. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv*.

Bolaños, M., Dimiccoli, M., y Radeva, P. (2015). Towards storytelling from visual lifelogging: An overview.
doi: 10.1109/THMS.2016.2616296

Catalán, J. M., diez, J., Blanco, A., Ezquerro, S., Barios, J., Badesa, F., y Garcia, N. (2017, 06). Aide: Adaptive multimodal interfaces to assist disabled people in daily activities grupo de investigación en neuroingeniería biomédica, universidad miguel hernández..

Dalal, N., y Triggs, B. (2005). Histograms of oriented gradients for human detection. En *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. IEEE. doi: 10.1109/cvpr.2005.177

Damen, D., Doughty, H., Farinella, G. M., Fidler, S., Furnari, A., Kazakos, E., … Wray, M. (2020). The epic-kitchens dataset: Collection, challenges and baselines.

david8862. (2020).
Descargado de `https://github.com/david8862/keras-YOLOv3-model-set`

Desai, S., Mantha, S. S., y Phalle, V. M. (2017, jan). Advances in smart wheelchair technology. En *2017 international conference on nascent technologies in engineering (ICNTE)*. IEEE. doi: 10.1109/icnte.2017.7947914

Fang, B., Co, J., y Zhang, M. (2017, nov). DeepASL. En *Proceedings of the 15th ACM conference on embedded network sensor systems*. ACM. doi: 10.1145/3131672.3131693

Girshick, R. (2015). Fast r-cnn.

Girshick, R., Donahue, J., Darrell, T., y Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation.

In, H., Kang, B. B., Sin, M., y Cho, K. (2015). Exo-glove: A wearable robot for the hand with a soft tendon routing system. *IEEE Robotics Automation Magazine*, *22*(1), 97-105. doi: 10.1109/mra.2014.2362863

Kang, B. B., Choi, H., Lee, H., y Cho, K.-J. (2019, apr). Exo-glove poly II: A polymer-based soft wearable robot for the hand with a tendon-driven actuation system. *Soft Robotics*, *6*(2), 214–227. doi: 10.1089/soro.2018.0006

Lindeberg, T. (2012). Scale invariant feature transform. *Scholarpedia*, *7*(5), 10491. doi: 10.4249/scholarpedia.10491

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., y Berg, A. C. (2015). Ssd: Single shot multibox detector.
doi: 10.1007/978-3-319-46448-0_2

Nguyen, T.-H.-C., Nebel, J.-C., y Florez-Revuelta, F. (2016, jan). Recognition of activities of daily living with egocentric vision: A review. *Sensors*, *16*(1), 72. doi: 10.3390/s16010072

Pirsiavash, H., y Ramanan, D. (2012, jun). Detecting activities of daily living in first-person camera views. En *2012 IEEE conference on computer vision and pattern recognition.* IEEE. doi: 10.1109/cvpr.2012.6248010

qqwweee. (2018).
Descargado de `https://github.com/qqwweee/keras-yolo3`

Randazzo, L., Iturrate, I., Perdikis, S., y d. R. Millan, J. (2018, jan). mano: A wearable hand exoskeleton for activities of daily living and neurorehabilitation. *IEEE Robotics and Automation Letters*, *3*(1), 500–507. doi: 10.1109/lra.2017.2771329

Redmon, J. (2013–2016). *Darknet: Open source neural networks in c.* `http://pjreddie.com/darknet/`.

Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2015). You only look once: Unified, real-time object detection.

Redmon, J., y Farhadi, A. (2016). Yolo9000: Better, faster, stronger.

Redmon, J., y Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv*.

Ren, S., He, K., Girshick, R., y Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks.

Ryoo, M. S., y Matthies, L. (2013, jun). First-person activity recognition: What are they doing to me? En *2013 IEEE conference on computer vision and pattern recognition.* IEEE. doi: 10.1109/cvpr.2013.352

Tan, M., y Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks.

Tan, M., Pang, R., y Le, Q. V. (2019). Efficientdet: Scalable and efficient object detection.

Viola, P., y Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. En *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001.* IEEE Comput. Soc. doi: 10.1109/cvpr.2001.990517